Sounding the Unknown: How Helical Sonification Unites Math, Music, and Emotion

by ChatGPT o1 ,

OpenAI Deep Research ,

and Nir Strulovitz

"Music is the pleasure the human mind experiences from counting without being aware that it is counting."

— Gottfried Wilhelm Leibniz

"Mathematics is the music of reason."

— James Joseph Sylvester

"Where words fail, music speaks."

— Hans Christian Andersen

From the book's back-cover:

What if you could **hear** an equation the same way you **see** a graph? In this groundbreaking fusion of science and art, the authors introduce the **Helical Sonification System (HSS)**, a revolutionary method that translates data into music-like structures. Combining **psychoacoustics**, **music theory**, and **data science**, they show how pitch, rhythm, and timbre can unveil hidden patterns in everything from mathematical functions to real-world time-series. Along the way, you'll discover:

- How a **helix** of pitch merges the octave's cyclical nature with continuous frequency rise,
- Why dissonant tones or sudden pitch glides can trigger primal fear or excitement,
- How polyrhythms and emotional "fear factors" can transform dry data into a heart-thumping immersive experience,
- Real-world case studies—from fractal curves to financial data—demonstrating how sonic mapping can sharpen insight or kindle creativity.

Whether you're a mathematician curious about new vistas, a musician seeking to push sonic boundaries, or a general reader fascinated by the unity of art and science, **Sounding the Unknown** offers an inspiring vision: a future where numbers truly **sing** and data becomes a grand symphony of discovery.

Sounding the Unknown: How Helical Sonification Unites Math, Music, and Emotion

by ChatGPT o1 , OpenAI Deep Research , and Nir Strulovitz

Preface

There's a special thrill in hearing something we usually only see or think about abstractly. We humans rely heavily on sight, yet a great deal of the universe's secrets—and even the secrets of our own data—remain hidden in numbers, equations, and invisible patterns. What if we could **hear** those patterns? What if a swirling fractal or a real-time data set could be perceived not just as a plot or a spreadsheet but as **music**? This book—Sounding the Unknown: How Helical Sonification Unites Math, Music, and Emotion—is about that transformative idea.

In the 1600s, René Descartes laid out a coordinate system that let mathematicians and physicists "see" equations in a plane. That moment paved the way for calculus and modern science. Today, we propose an analogous leap: a **sonic** coordinate system, weaving together pitch, timbre, rhythm, and emotion to represent data in a musical dimension. We call it the "Helical Sonification System" (HSS), because at its core is a **helix** that captures the cyclical nature of pitch classes and the linear rise of frequency. But that's just our starting point.

This book tries to do two things. First, it introduces you to a practical method for mapping numbers—functions, time-series, multi-dimensional data—onto a music-like structure. Along the way, we'll see how psychoacoustics, music theory, and cognitive psychology come together. You'll learn about roughness and dissonance, polyrhythms, micro-timbral changes, and even emotional shading: how adding a "fear factor" can turn a neutral pitch line into a chilling alarm that raises your heart rate.

Second, we want to convey a sense of **wonder** and **possibility**. Historically, once we had Cartesian coordinates, we could apply powerful tools like calculus to solve problems that changed the world. Similarly, by giving equations and data **audible** form, we hope to unlock new forms of insight. Imagine a scientist in a VR environment listening for anomalies in a high-dimensional dataset, or a medical researcher tracking

subtle EEG patterns by ear, or an artist merging fractal geometry with eerie modulated sirens to create a haunting new composition.

This book is both **technical** and **visionary**. You'll find pages of code and references to real psychoacoustic studies. If you're a musician, data scientist, game audio engineer, or just an enthusiast intrigued by how math and music intertwine, we hope there's something for you here. You can dive into code examples in Python notebooks or skip to the conceptual discussions on how pitch glides, dissonance intervals, or polyrhythms evoke primal emotions.

In short, we invite you to think of data not as "dry numbers," but as a potential **symphony** waiting to be orchestrated. Just as a student might first see an equation and learn to plot it, we want to empower you to **listen** to that equation, gleaning patterns your eyes might miss. This is our hope for "Sounding the Unknown": a step toward a future where data, mathematics, emotion, and sonic artistry merge into a new dimension of understanding. Thank you for joining us on this journey—let's tune in and see what the data has to say.

Helical Sonification: A Unified Approach to Mapping Equations into Music

ChatGPT o1, DeepSeek R1, OpenAI Deep Research, and Nir Strulovitz

February 9, 2025

Contents

Chapter 1

Introduction and Motivation

1.1 Motivation

A brief history of sonification attempts, illustrating why naive "function playback" often fails to produce musically coherent or intuitive results. Explanation of why a *helical coordinate system* for pitch can capture octave equivalences (cyclic) while maintaining continuous pitch height.

1.2 Scope

Clarifying that this book focuses on **one** integrated method rather than surveying many alternatives. Overview of the "three-axis + optional expansions" model:

- Pitch (helix)
- Timbre (one or more parameters)
- Time (musical timeline)
- (Optional) Extra dimension (e.g. rhythm/polyrhythm, emotional shading)

1.3 Chapter Preview

Short summary of each upcoming chapter and its goals, with a promise that each chapter will include **practical code examples in Python**.

1.4 Prerequisites

Discuss what the reader needs:

- Basic music theory knowledge (intervals, scales, octaves).
- Familiarity with Python (installing packages, running scripts).

Chapter 2

Mathematical and Musical Foundations

2.1 Recap of Basic Music Theory

Pitch classes, octaves, scales, chords, intervals, and how they tie into sonification.

2.2 Why a Helix for Pitch?

Explanation of cyclic vs. linear aspects of pitch, referencing Shepard's pitch helix or similar spiral models.

2.3 Core Math Concepts

Log-frequency, group structures (very lightly), coordinate systems for pitch.

2.4 Simple Code Demo (No Audio Yet)

Minimal Python code to generate or visualize a helix of semitones, highlighting the difference between purely linear vs. spiral pitch mapping.

Chapter 3

Implementing the Helical Pitch Axis

3.1 Coordinate Definition

Deriving and explaining

$$x(\theta) = r\cos(\theta), \quad y(\theta) = r\sin(\theta), \quad z(\theta) = k\,\theta,$$

where θ could correspond to semitones.

3.2 Discrete vs. Continuous Steps

Balancing the need for scale quantization versus continuous sweeps in pitch.

3.3 Hands-On Code

Python script that plots and optionally plays discrete notes along a helical pitch axis.

3.4 Avoiding Siren-Like Sounds

Discussion of scale quantization, rhythmic subdivisions, and envelope shaping.

Chapter 4

Timbre as a Second Dimension

4.1 Understanding Timbre

Brief psychoacoustic notes: spectral centroid, brightness, attack time. Emphasizing an approach that's simple enough to remain musical.

4.2 One-Dimensional Timbre Parameter

Deciding on a single parameter for brightness or filter cutoff as the main axis.

4.3 Implementation

Sample Python code, possibly using `pyaudio` or another library to demonstrate real-time or offline generation. Explaining how to combine pitch (helix) with timbre parameter changes.

4.4 Example Project

A small demonstration that plays ascending notes along the pitch helix while linearly transitioning from "dark" timbre to "bright."

Chapter 5

Practical Sonification of Mathematical Functions

5.1 General Mapping Scheme

$$\begin{aligned} x &\mapsto \text{time / discrete steps}, \\ f(x) &\mapsto \text{pitch (helix)}, \\ \text{possibly } \frac{df}{dx}, \ldots &\mapsto \text{timbre}. \end{aligned}$$

5.2 Avoiding Common Pitfalls

Preventing continuous sweeps, dealing with large function ranges, smoothing data if needed.

5.3 Example: Damped Sine

$$f(x) = e^{-x} \sin(10x).$$

Mapping to pitch + timbre in Python, with code to illustrate the approach.

5.4 Comparison

Illustrate how naive function playback differs drastically from the *helical sonification* approach.

Chapter 6

Rhythm, Meter, and Polyrhythms (Optional)

6.1 Why Rhythm Matters

Time as another fundamental axis, and polyrhythm as a potential encoding of extra dimensions.

6.2 Implementation

How to incorporate polyrhythms or variable note durations in Python. Possibly using thresholds in the data to trigger different rhythmic patterns.

6.3 Short Example

Implementation of a 3:4 polyrhythm demonstration, mapping data to separate rhythmic tracks.

Chapter 7

Emotional Shading (Optional)

7.1 Brief Theoretical Note

Acknowledging Hevner's circle, Russell's circumplex, or other emotional models.

7.2 Practical Minimalism

Mapping something like "positive vs. negative" to major vs. minor scale or chord expansions, or adjusting dynamic range as a stand-in for arousal.

7.3 Implementation Example

Show a snippet that toggles the pitch set (major/minor) based on a user-defined emotional parameter.

Chapter 8

Advanced Topics and Scaling Up

8.1 Multi-Voice Sonification

Assign multiple functions or data streams to separate instruments or timbral layers.

8.2 Large Data Sets

Approaches for handling time-series data in Python, slicing it into manageable chunks.

8.3 Real-Time Interaction

Using libraries like `sounddevice` (Python) for real-time sonification, possibly with a simple GUI or sliders for user control.

Chapter 9

Practical Applications and Case Studies

9.1 Physics Example

Implementing the Brachistochrone or Planetary Orbits in code, converting them into a coherent helical sonification.

9.2 Data Visualization Example

Stock market or polling data turned into sonification, complete with code for reading CSV, normalizing data, and mapping it to pitch/timbre.

9.3 Mathematical Shapes

Mapping parametric curves (e.g., Lissajous, fractals) to the system, exploring how each dimension translates into a sonic gesture.

Chapter 10

Conclusion and Future Directions

10.1 Summary of the Helical Sonification System

Recap the benefits of a pitch helix, a timbre axis, and thoughtful rhythmic structures.

10.2 Extensions

Brainstorm on microtonal expansions, emotional complexity, VR-based or gaming integrations.

10.3 Final Thoughts

Reiterate the central achievement: a single, musically coherent approach to sonification, with code examples that readers can immediately try and modify.

Chapter 1: Introduction and Motivation

Preliminary Concepts

February 9, 2025

1.1 Motivation

Sonification—the process of translating data or mathematical structures into sound—has been explored in many scientific and artistic contexts. However, many attempts produce audio that is more akin to beeps, continuous sweeps, or "vacuum cleaner" noises rather than anything a listener would recognize as *musical*.

Common Shortcomings of Simple Sonification

- **Raw Function Playback:** Converting a function $f(t)$ directly to a waveform (e.g., using Mathematica's `Play[]`) leads to continuous pitch sweeps or droning timbres.
- **No Musical Framework:** Data is not quantized to scales, chords, or rhythms, so the ear cannot latch onto familiar patterns.
- **Limited Usefulness:** Without musical scaffolding, sonified data is often unintuitive and frustrating to interpret.

A more robust approach involves defining a **coordinate system for sound** that parallels Cartesian coordinates for geometry. This coordinate system must accommodate:

- **Pitch**, reflecting how humans perceive frequency in both its cyclic (octave) and linear (continuous) aspects.
- **Timbre**, since changes in the harmonic spectrum or brightness can convey vital dimensions of information.
- **Time** or **rhythm**, anchoring the sonic display in a perceptually meaningful temporal grid.

1.2 Scope and Goals

Focusing on One Integrated Method

In this book, we will not survey every possible sonification strategy. Instead, we concentrate on a single, unified **helical sonification system (HSS)** that:

- Maps data to pitch in a **helical** structure, capturing octave equivalences while ascending in frequency.
- Allows for a **timbre axis** or parameter to represent additional data variation in a perceptually meaningful way.
- Places notes in a **musical timeline**, optionally using rhythms or polyrhythms to encode further dimensions.

This approach transforms raw data into something the ear recognizes as music-like, promoting intuitive insight much like how Cartesian coordinates help us "see" equations.

Key Objectives

- **Establish a Practical Framework:** Each chapter will introduce theory followed by concrete Python code.
- **Bridge Math and Music:** Build an understanding of how to use pitch, timbre, and time to map data sets or analytical functions into coherent musical structures.
- **Enable Immediate Experimentation:** Offer code snippets that let readers generate audio and experiment with parameters in near real-time.

1.3 Chapter Preview

The book is structured so that each chapter adds one key layer of our helical sonification method:

- **Chapter 2: Mathematical & Musical Foundations**
  Recaps basic music theory (pitch, scales, chords) and the essential math for our helical pitch model.
- **Chapter 3: Implementing the Helical Pitch Axis**
  Introduces the explicit helix equation and shows how to discretize frequencies into musical pitches.
- **Chapter 4: Timbre as a Second Dimension**
  Explains how to incorporate a single timbre parameter, such as brightness, into the sonification coordinate system.

- **Chapter 5: Practical Sonification of Functions**
  Demonstrates mapping typical math/physics functions onto pitch and timbre with real code examples.
- **Chapter 6: Rhythm, Meter, and Polyrhythms (Optional)**
  Explores how to use rhythm or polyrhythm as an additional dimension if desired.
- **Chapter 7: Emotional Shading (Optional)**
  Adds a minimal approach for emotional cues (e.g., major vs. minor, dynamic levels).
- **Chapters 8–9: Advanced Topics and Applications**
  Covers multi-voice sonification, large data sets, real-time interactive setups, and case studies (e.g., Brachistochrone curves, stock market data).
- **Chapter 10: Conclusion and Future Directions**
  Summarizes the method, limitations, and possible expansions.

1.4 Prerequisites

Minimal Music Theory

This project assumes a basic understanding of:

- **Pitch Classes and Octaves:** Recognizing that C in one octave is "the same note" but at a higher frequency than C in a lower octave.
- **Scales and Chords:** Familiarity with at least one or two scales (e.g., major, minor) and triads.

Basic Python Familiarity

We will provide code examples in Python. The reader should be able to:

- Install and import libraries (e.g., `numpy`, `matplotlib`, `sounddevice` or similar).
- Run Python scripts and tweak parameters.
- Optionally, create simple GUIs or interactive sliders (using libraries like `ipywidgets` in Jupyter notebooks).

1.5 Conclusion

By the end of this book, you will have a working **helical sonification system** that can take any suitable numerical data set or mathematical function and generate music-like audio. This method aims to provide both scientific insight and creative exploration, enabling you to "hear" patterns, trends, and relationships that might otherwise remain abstract.
Next, we delve into the foundations of music theory and the essential mathematics behind our helical pitch axis.

Chapter 1: Prerequisites with Conda Setup (Revised)

February 9, 2025

Prerequisites (Revised)

This project uses **Python** and **Jupyter Notebook** (or JupyterLab) to provide interactive code examples for sonification. Below is a step-by-step guide to installing these tools using **Anaconda**, which is a popular Python distribution that simplifies environment management.

Why Conda/Anaconda? Using `conda` environments ensures that all packages are installed in one coherent, isolated setup, so you won't run into `ModuleNotFoundError` issues from mismatched Python installs.

1. Download and Install Anaconda

1. Visit `https://www.anaconda.com/products/distribution` and download the latest Anaconda installer for Windows (64-bit).
2. Run the installer. During the setup, check the option to "Add Anaconda to my PATH environment variable" if prompted (if you're comfortable with that) or note that you can use the "Anaconda Prompt" to manage everything.
3. Once installation finishes, you should have an "Anaconda Prompt" available in your Start Menu (on Windows).

2. Create a New Conda Environment

1. Open the **Anaconda Prompt** (look in your Start Menu for "Anaconda3 (64-bit) ¿ Anaconda Prompt").
2. Type:

   ```
   conda create --name sonify_env python=3.9
   ```

   ("sonify_env" is just a name; you can choose another. Here we use Python 3.9, which is well-supported by most libraries.)
3. After it asks you to proceed, type 'y' and press Enter. This will create a new environment with the name `sonify_env`.

3. Activate Your Environment

1. In the Anaconda Prompt, type:

```
conda activate sonify_env
```

2. You should see `(sonify_env)` appear in your command prompt, indicating that your new environment is active.

4. Install Jupyter and Required Packages

Now install everything we need in `sonify_env`:

1. **Install Jupyter Notebook or JupyterLab:**

```
conda install jupyter
```

Or if you prefer JupyterLab:

```
conda install jupyterlab
```

2. **Install Additional Python Packages:**

```
conda install numpy
conda install -c conda-forge python-sounddevice
conda install ipywidgets
```

(We use the conda-forge channel for `python-sounddevice` because it's well-maintained there.)

3. **Optional** (if you want to do quick plotting):

```
conda install matplotlib
```

This ensures that `numpy`, `sounddevice`, `ipywidgets`, and Jupyter are all in the same environment and accessible.

5. Launch Jupyter Notebook

1. While still in `(sonify_env)`, type:

   ```
   jupyter notebook
   ```

   or

   ```
   jupyter lab
   ```

2. A web browser window should open, showing the Jupyter interface. Navigate to the folder where you keep your notebooks or code.

5.1 Verifying `sounddevice`

Create a quick test notebook. In a new code cell, try:

```
import sounddevice as sd
import numpy as np

samplerate = 44100
duration = 1.0
frequency = 440.0  # A4
t = np.linspace(0, duration, int(samplerate * duration), endpoint=False)
waveform = 0.3 * np.sin(2 * np.pi * frequency * t)

sd.play(waveform, samplerate=samplerate)
sd.wait()
```

If you hear a short beep, `sounddevice` is working.

Potential Troubleshooting

- **No Sound?** Make sure your speakers/headphones are on and the volume is sufficient. Also check Windows audio settings or drivers.
- **ModuleNotFoundError?** Double-check that you have "(sonify_env)" visible in the prompt and that you installed the library *inside* that environment.
- **Multiple Python Versions?** If you had other Python installs before, they should not interfere as long as you're working from Anaconda Prompt with the correct activated environment.

Additional Notes

This environment setup is recommended for all the code examples throughout the book. We will provide scripts and notebooks that use `numpy`, `sounddevice`, and `ipywidgets` for interactive sonification demos. By consistently activating `sonify_env`, you should avoid almost all version or dependency conflicts.

Enjoy the sonification examples! From now on, whenever the book references a Python code snippet, open your Anaconda Prompt, `conda activate sonify_env`, and launch Jupyter to ensure you have the correct environment.

Chapter 1: Introduction and Motivation
Expanded Discussion

February 9, 2025

1.1 Historical Context of Sonification

Sonification has been explored in various forms for centuries, often intersecting with artistic or scientific curiosities:

- **Early Experiments (17th–19th Centuries).** Scholars like *Marin Mersenne* (1588–1648) and *Isaac Newton* (1642–1726/7) toyed with linking musical pitches to planetary motion or color spectra. However, these were largely speculative attempts with no formalized notion of "sonifying data" as we understand it today.
- **20th Century Explorations.** Composers such as *Iannis Xenakis* and *Lejaren Hiller* used mathematical structures (probability distributions, stochastic processes) to generate music. Although these works were more *composition* than *data mapping*, they hinted at the potential for bridging numeric representations and sound.
- **Modern Data Sonification.** Since the 1980s, scientists and musicians have tried turning scientific data (e.g., seismic readings, stock prices, brainwave signals) into audio. NASA famously sonified astronomical data sets (like black hole acoustic signatures or pulsar signals) to raise public awareness and enable alternative analyses.

Despite these endeavors, many first-generation sonification projects suffer from a lack of musical scaffolding, leading to raw, non-intuitive signals.

1.2 Common Pitfalls and Lessons Learned

Lack of Musical Scaffolding

- Directly mapping numeric values to continuous pitch results in *glissandi* or sweeps that are perceived as siren-like or irritating drones.
- Without rhythmic structure, such sonifications feel arbitrary in timing and fail to convey *pulse* or *groove*, which are central to human musical perception.

Overloaded Timbre

- Some sonification attempts attempt to encode too many parameters in timbre (e.g., modulating waveform shape, brightness, stereo panning). This can overwhelm listeners and obscure meaningful patterns.
- Simplicity is key: focusing on one or two timbre parameters (e.g., spectral centroid, attack) helps maintain a clear sonic identity.

No Clear Mapping Logic

- When changes in the data lack a clear perceptual correlation (e.g., random micro-changes in amplitude or pitch), listeners cannot form stable *mental representations* of the data relationships.
- A well-defined coordinate system clarifies how each dimension of the data (e.g., function value, derivative, category) maps to an audible parameter (pitch, timbre, volume, rhythm).

1.3 Why a Coordinate-Based Approach?

Visual graphs revolutionized mathematics by linking algebraic formulas with *geometry*, enabling us to "see" shapes of functions, intersections, and symmetries. Our approach tries to replicate that power in the *auditory* domain:

- **Spatial Analogy:** Just as Cartesian (x, y) coordinates let us plot a curve, our *helical* pitch-timbre-time coordinate system provides an auditory "space" to move within.
- **Helical Axis for Pitch:** Humans perceive pitch cyclically (octaves) yet also sense continuous ascending or descending relationships. A helix naturally merges these two aspects.
- **Timbre Axis:** Another dimension can represent the spectral shape of the sound (brightness, harmonic content) in a way that is easier to track than raw waveforms.
- **Time Grid:** Organizing sounds into discrete beats or rhythmic subdivisions helps us perceive data changes as musically meaningful events rather than random bleeps.

This structure fosters *musical* intuition: repeated intervals, octave equivalences, chord-like sonorities, and rhythmic motifs become recognizable, so data patterns are heard as variations in melody or texture.

1.4 Vision for the Helical Sonification System (HSS)

Connecting Math and Music

We adopt the concept of a *helix* for pitch to unify cyclic and linear pitch perception. Each full turn of the helix represents one octave; moving vertically up the helix (the z-axis) corresponds to rising pitch frequency in a continuous manner.

Adding Timbre to the Picture

We reserve an extra dimension (or axis) for timbre, typically simplified to one parameter (like "brightness" or filter cutoff) to avoid psychoacoustic overload. This ensures that timbre changes reinforce or clarify data variations rather than obscuring them.

Discrete Time for Musicality

Time is crucial for organizing sonification into a *musical timeline*. Instead of mapping data to raw time increments, we slice time into beats or measures, letting the ear parse patterns as rhythmic phrases.

1.5 Who Can Benefit From HSS?

- **Scientists and Researchers:** May discover patterns in high-dimensional data that are less obvious visually, or they want to present complex phenomena (e.g., astrophysical processes, neural signals) in an engaging way.
- **Educators:** Teaching mathematical concepts (like functions, derivatives, or even advanced geometry) through *auditory* analogies can aid students who struggle with purely visual or symbolic approaches.
- **Musicians and Composers:** Looking for new creative tools to integrate data into compositions or interactive installations, bridging science and art.

1.6 Additional References and Inspirations

Below are a few references that inform or inspire the helical sonification approach:

- **R. N. Shepard (1964)**. "Circularity in Judgments of Relative Pitch," *Journal of the Acoustical Society of America*. Introduces the concept of a pitch helix, blending octave equivalences with continuous pitch height.
- **Elaine Chew (2014)**. *Mathematical and Computational Modeling of Tonality*, Springer. Discusses spiral array models for pitch, chords, and keys in 3D spaces.
- **Diana Deutsch (Ed., 2012)**. *The Psychology of Music*, 3rd ed., Academic Press. Explores psychoacoustic phenomena and how humans perceive pitch and timbre.

- **Carla Scaletti (1988–present)**. A pioneer in sonification and interactive music software (*Kyma* system), emphasizing the importance of mapping data to musical parameters with user control.
- **Iannis Xenakis (1971)**. *Formalized Music*: Explores the intersection of mathematics, stochastic processes, and composition, an early blueprint for data-driven music.

1.7 Conclusion

In this expanded view of Chapter 1, we see that *musical sonification* requires balancing mathematical fidelity with psychoacoustic realities. The helical sonification system (HSS) aims to provide a coherent coordinate-based structure that:

1. Captures octave equivalences while allowing ascending pitch lines.
2. Uses timbre meaningfully without overwhelming the listener.
3. Places data events in a rhythmic framework, creating a musically intelligible timeline.

Armed with these insights, we proceed to the next chapter, which covers the essential mathematical concepts and fundamental music theory needed to implement HSS. There, we will set the stage for translating raw data into pitch, timbre, and time coordinates in a consistent, listener-friendly way.

Next, we will delve into Chapter 1's code examples (Part 3), where you'll see a simple interactive Python setup for playing with naive vs. structured sonification.

naive_playback.py

```
import numpy as np
import sounddevice as sd

def naive_playback(f0=220, slope=100, duration=2.0, fs=44100):
    """
    Play a continuously sliding tone:
    frequency(t) = f0 + slope * t.
    """
    t = np.linspace(0, duration, int(fs*duration), endpoint=False)
    freq = f0 + slope * t
    # Generate sine wave
    wave = np.sin(2.0 * np.pi * freq * t)
    sd.play(wave, samplerate=fs)
    sd.wait()  # wait until playback finishes

if __name__ == "__main__":
    # Example usage
    naive_playback(f0=220, slope=100, duration=3.0)
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Interactive Slider Example\n",
    "This snippet uses `ipywidgets` to allow the user to control the slope parameter in `naive_playback`."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {},
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "613924e3a1cd428d9b451090de9b6fa4",
       "version_major": 2,
       "version_minor": 0
      },
      "text/plain": [
       "FloatSlider(value=100.0, continuous_update=False, description='Slope:', max=300.0, step=1.0)"
      ]
     },
     "metadata": {},
```

"output_type": "display_data"
},
{
"data": {
"application/vnd.jupyter.widget-view+json": {
"model_id": "0f8307e3924845259c4d8ddbc02fe1b9",
"version_major": 2,
"version_minor": 0
},
"text/plain": [
"Button(description='Play', style=ButtonStyle(), tooltip='Play siren with current slope')"
]
},
"metadata": {},
"output_type": "display_data"
}
],
"source": [
"import numpy as np\n",
"import sounddevice as sd\n",
"import ipywidgets as widgets\n",
"from IPython.display import display\n",
"\n",
"def naive_playback(f0=220, slope=100, duration=2.0, fs=44100):\n",
"   t = np.linspace(0, duration, int(fs*duration), endpoint=False)\n",
"   freq = f0 + slope * t\n",
"   wave = np.sin(2.0 * np.pi * freq * t)\n",

```
    "    sd.play(wave, samplerate=fs)\n",
    "    sd.wait()\n",
    "\n",
    "# Create a slider for slope\n",
    "slope_slider = widgets.FloatSlider(\n",
    "    value=100,\n",
    "    min=0,\n",
    "    max=300,\n",
    "    step=1.0,\n",
    "    description='Slope:',\n",
    "    continuous_update=False\n",
    ")\n",
    "\n",
    "# Create a button to trigger playback\n",
    "play_button = widgets.Button(\n",
    "    description='Play',\n",
    "    button_style='',\n",
    "    tooltip='Play siren with current slope'\n",
    ")\n",
    "\n",
    "def on_button_click(b):\n",
    "    naive_playback(slope=slope_slider.value)\n",
    "\n",
    "play_button.on_click(on_button_click)\n",
    "\n",
    "# Display the slider and button\n",
    "display(slope_slider, play_button)"
   ]
  },
```

```
{
 "cell_type": "code",
 "execution_count": 1,
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "C:\\Python313\\python.exe\n"
   ]
  }
 ],
 "source": [
  "import sys\n",
  "print(sys.executable)"
 ]
},
{
 "cell_type": "code",
 "execution_count": 1,
 "metadata": {},
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "C:\\Users\\nir_s\\anaconda3\\envs\\sonify_env\\python.exe\n"
   ]
```

```
   }
  ],
  "source": [
   "import sys\n",
   "print(sys.executable)"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": null,
  "metadata": {},
  "outputs": [],
  "source": []
 }
],
"metadata": {
 "kernelspec": {
  "display_name": "Sonify Env",
  "language": "python",
  "name": "sonify_env"
 },
 "language_info": {
  "codemirror_mode": {
   "name": "ipython",
   "version": 3
  },
  "file_extension": ".py",
  "mimetype": "text/x-python",
  "name": "python",
```

```
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.21"
  },
  "name": "interactive_slider"
 },
 "nbformat": 4,
 "nbformat_minor": 4
}
```

structured_sonification.py

```python
import numpy as np
import sounddevice as sd

# A simple C major scale in semitones
SCALE = [0, 2, 4, 5, 7, 9, 11]

def nearest_pitch(value):
    """
    Snap a real number value to the nearest semitone in SCALE.
    For example, if value ~ 3.2, this returns 4.
    """
    # Round to nearest integer
    base_semitone = round(value)
    # We then find the offset in a standard 12-tone set
    offset = base_semitone % 12
    # Find the closest note in the SCALE list
    closest = min(SCALE, key=lambda x: abs(x - offset))
    # This recovers the full absolute semitone
    final_semitone = base_semitone - offset + closest
    return final_semitone

def structured_sonification(func, x_start=0.0, x_end=5.0, steps=25,
                            base_note=60, dur_per_note=0.3, fs=44100):
    """
    Convert a function f(x) to discrete pitches in a scale,
    and play short notes at each pitch.
    :param func: A Python function f(x) returning some real value
    :param base_note: MIDI note offset (60 = Middle C)
    """
    x_vals = np.linspace(x_start, x_end, steps)
    # We'll accumulate audio in a buffer, then play once
    note_samples = []

    for x in x_vals:
        val = func(x)
        # Map val -> semitone offset
        semitone = nearest_pitch(val)
        midi_note = base_note + semitone
        freq = 440.0 * 2**((midi_note - 69)/12.0)  # MIDI->Hz

        t = np.linspace(0, dur_per_note, int(fs*dur_per_note), endpoint=False)
        wave = 0.1 * np.sin(2.0 * np.pi * freq * t)  # amplitude = 0.1
        note_samples.append(wave)

    # Concatenate all notes
    full_wave = np.concatenate(note_samples)
    sd.play(full_wave, samplerate=fs)
    sd.wait()

if __name__ == "__main__":

    def example_func(x):
        # e^(-x) sin(10x) as an example
        return np.exp(-x)*np.sin(10*x)*12  # scaled by 12 for semitone range

    structured_sonification(example_func, x_end=6.0, steps=30)
```

```python
import numpy as np
import sounddevice as sd

# A simple C major scale in semitones
SCALE = [0, 2, 4, 5, 7, 9, 11]

def nearest_pitch(value):
    """
    Snap a real number value to the nearest semitone in SCALE.
    For example, if value ~ 3.2, this returns 4.
    """
    # Round to nearest integer
    base_semitone = round(value)
    # We then find the offset in a standard 12-tone set
    offset = base_semitone % 12
    # Find the closest note in the SCALE list
    closest = min(SCALE, key=lambda x: abs(x - offset))
    # This recovers the full absolute semitone
    final_semitone = base_semitone - offset + closest
    return final_semitone

def structured_sonification(func, x_start=0.0, x_end=5.0, steps=25,
                            base_note=60, dur_per_note=0.3,
fs=44100):
    """
    Convert a function f(x) to discrete pitches in a scale,
    and play short notes at each pitch.
    :param func: A Python function f(x) returning some real value
    :param base_note: MIDI note offset (60 = Middle C)
```

```
    """
    x_vals = np.linspace(x_start, x_end, steps)
    # We'll accumulate audio in a buffer, then play once
    note_samples = []

    for x in x_vals:
        val = func(x)
        # Map val -> semitone offset
        semitone = nearest_pitch(val)
        midi_note = base_note + semitone
        freq = 440.0 * 2**((midi_note - 69)/12.0)  # MIDI->Hz

        t = np.linspace(0, dur_per_note, int(fs*dur_per_note),
endpoint=False)
        wave = 0.1 * np.sin(2.0 * np.pi * freq * t)  # amplitude =
0.1
        note_samples.append(wave)

    # Concatenate all notes
    full_wave = np.concatenate(note_samples)
    sd.play(full_wave, samplerate=fs)
    sd.wait()

if __name__ == "__main__":

    def example_func(x):
        # e^(-x) sin(10x) as an example
        return np.exp(-x)*np.sin(10*x)*12  # scaled by 12 for
semitone range
```

```
structured_sonification(example_func, x_end=6.0, steps=30)
```

interactive_slider.ipynb

```
#%% md
# Interactive Slider Example
This snippet uses `ipywidgets` to allow the user to control the slope parameter in
`naive_playback`.
#%%
import numpy as np
import sounddevice as sd
import ipywidgets as widgets
from IPython.display import display

def naive_playback(f0=220, slope=100, duration=2.0, fs=44100):
    t = np.linspace(0, duration, int(fs*duration), endpoint=False)
    freq = f0 + slope * t
    wave = np.sin(2.0 * np.pi * freq * t)
    sd.play(wave, samplerate=fs)
    sd.wait()

# Create a slider for slope
slope_slider = widgets.FloatSlider(
    value=100,
    min=0,
    max=300,
    step=1.0,
    description='Slope:',
    continuous_update=False
)

# Create a button to trigger playback
play_button = widgets.Button(
    description='Play',
    button_style='',
    tooltip='Play siren with current slope'
)

def on_button_click(b):
    naive_playback(slope=slope_slider.value)

play_button.on_click(on_button_click)

# Display the slider and button
display(slope_slider, play_button)
#%%
import sys
print(sys.executable)
#%%
import sys
print(sys.executable)
#%%
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Interactive Slider Example\n",
    "This snippet uses `ipywidgets` to allow the user to control the
slope parameter in `naive_playback`."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "613924e3a1cd428d9b451090de9b6fa4",
       "version_major": 2,
```

```
        "version_minor": 0
       },
       "text/plain": [
        "FloatSlider(value=100.0, continuous_update=False, 
description='Slope:', max=300.0, step=1.0)"
       ]
      },
      "metadata": {},
      "output_type": "display_data"
     },
     {
      "data": {
       "application/vnd.jupyter.widget-view+json": {
        "model_id": "0f8307e3924845259c4d8ddbc02fe1b9",
        "version_major": 2,
        "version_minor": 0
       },
       "text/plain": [
        "Button(description='Play', style=ButtonStyle(), 
tooltip='Play siren with current slope')"
       ]
      },
      "metadata": {},
      "output_type": "display_data"
     }
    ],
    "source": [
     "import numpy as np\n",
     "import sounddevice as sd\n",
```

```
    "import ipywidgets as widgets\n",
    "from IPython.display import display\n",
    "\n",
    "def naive_playback(f0=220, slope=100, duration=2.0,
fs=44100):\n",
    "    t = np.linspace(0, duration, int(fs*duration),
endpoint=False)\n",
    "    freq = f0 + slope * t\n",
    "    wave = np.sin(2.0 * np.pi * freq * t)\n",
    "    sd.play(wave, samplerate=fs)\n",
    "    sd.wait()\n",
    "\n",
    "# Create a slider for slope\n",
    "slope_slider = widgets.FloatSlider(\n",
    "    value=100,\n",
    "    min=0,\n",
    "    max=300,\n",
    "    step=1.0,\n",
    "    description='Slope:',\n",
    "    continuous_update=False\n",
    ")\n",
    "\n",
    "# Create a button to trigger playback\n",
    "play_button = widgets.Button(\n",
    "    description='Play',\n",
    "    button_style='',\n",
    "    tooltip='Play siren with current slope'\n",
    ")\n",
    "\n",
```

```
    "def on_button_click(b):\n",
    "    naive_playback(slope=slope_slider.value)\n",
    "\n",
    "play_button.on_click(on_button_click)\n",
    "\n",
    "# Display the slider and button\n",
    "display(slope_slider, play_button)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "C:\\Python313\\python.exe\n"
     ]
    }
   ],
   "source": [
    "import sys\n",
    "print(sys.executable)"
```

```
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "C:\\Users\\nir_s\\anaconda3\\envs\\sonify_env\\python.exe\n"
     ]
    }
   ],
   "source": [
    "import sys\n",
    "print(sys.executable)"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "pycharm": {
```

```
    "name": "#%%\n"
   }
  },
  "outputs": [],
  "source": []
 }
],
"metadata": {
 "kernelspec": {
  "display_name": "Sonify Env",
  "language": "python",
  "name": "sonify_env"
 },
 "language_info": {
  "codemirror_mode": {
   "name": "ipython",
   "version": 3
  },
  "file_extension": ".py",
  "mimetype": "text/x-python",
  "name": "python",
  "nbconvert_exporter": "python",
  "pygments_lexer": "ipython3",
  "version": "3.9.21"
 },
 "name": "interactive_slider"
},
"nbformat": 4,
"nbformat_minor": 4
```

```
}
```

Chapter 2: Mathematical and Musical Foundations
Preliminary Concepts

February 9, 2025

2.1 Overview

In the previous chapter, we emphasized the shortcomings of naive sonification and introduced the vision of a **helical sonification system (HSS)**. Before we can implement such a system, we need to establish the basics of:

- **Core Music Theory** (pitches, intervals, scales, octaves).
- **Mathematical Foundations** (log-frequency mappings, simple group concepts).

This preliminary discussion sets the stage for the detailed implementation of the pitch helix in later chapters. We keep this section concise; the *Expanded Discussion* later in Chapter 2 will dive deeper into each concept.

2.2 Basic Music Theory Concepts

Pitches and Octaves

A musical **pitch** is often measured in Hertz (cycles per second). However, human perception of pitch is roughly **logarithmic**: going from 220 Hz to 440 Hz (an octave) feels like "the same distance" as going from 440 Hz to 880 Hz. This is why log(frequency) mappings are so central to sonification.

Pitch Classes

Notes like C, D, E, etc. are often referred to as *pitch classes*, ignoring which octave they are in. For instance, C_4 (middle C) and C_5 are in different octaves but share the same pitch class (C).

Scales and Intervals

A **scale** is an ordered set of pitches. In Western music, the 12-tone *equal tempered* system is common, but you might use a subset (like a major scale) or a microtonal scale with more than 12 notes per octave.

- **Major Scale Example (C major):** C, D, E, F, G, A, B, C.
- **Intervals:** The distance between notes is measured in semitones (equal steps) in 12-TET. C to C$\sharp$ is 1 semitone, C to D is 2 semitones, etc.

Chords (Optional Preview)

A **chord** is when multiple pitches (often 3 or more) sound simultaneously. Later chapters will mention chords as a way to represent multidimensional data or to create richer sonic structures.

2.3 Mathematical Foundations

Log-Frequency for Pitch

Because we perceive pitch *intervals* in terms of *frequency ratios*, the mapping

$$p = \log_2(\text{frequency})$$

often makes sense. A difference of $+1$ in p means doubling the frequency (an octave). This explains why we visualize pitch as a vertical axis on a *log scale*.

Cyclic vs. Linear Aspects of Pitch

- **Cyclic (Octave Equivalence):** C_4 and C_5 are perceived as "the same note" in a higher register. So there's a repeating cycle (mod 12 semitones).
- **Linear (Continuous Ascent):** We also sense that C_5 is strictly higher than C_4. The *helix* merges these ideas by wrapping a circle for the pitch class while still ascending in height.

Simple Group Theoretic Idea

While we won't delve too deeply into group theory, it helps to note:

- The set of *pitch classes* can be seen as $\mathbb{Z}_{12}$ under addition (in 12-tone equal temperament).
- The concept of *octave equivalence* can be viewed as a quotient of real frequencies by factors of 2. In simpler terms, f and $2f$ share the same pitch class.

2.4 Helical Coordinates (Preview)

Why a Helix?

1. **Octave Equivalence:** We want each $2\times$ jump in frequency to loop back around to the same note name.

2. **Continuous Ascension:** We still need to differentiate one octave from the next (C_4 vs. C_5). A helix accomplishes both by letting each 2π rotation in angle correspond to a factor-of-2 increase in frequency.

Mathematically, we might parametrize:

$$\begin{cases} x(\theta) = r\cos(\theta), \\ y(\theta) = r\sin(\theta), \\ z(\theta) = k\,\theta, \end{cases}$$

where θ might correspond to semitones or continuous pitch changes. In Chapter 3, we'll implement this explicitly.

2.5 Putting It All Together

To summarize, we need:

- **A scale or pitch set**: so we can move in discrete (or microtonal) steps within each octave.
- **A log-based approach to frequency**: so the ear perceives intervals consistently in the sonification.
- **A helical model**: to represent both the cyclical (mod 12) and linear (octave to octave) nature of pitch.

This foundation will enable us to build a robust musical coordinate system for sonification. At the end of Chapter 2 (Expanded Discussion), we'll delve deeper into each concept, referencing some historical and theoretical works (Shepard's pitch helix, Chew's spiral array, etc.).

We will now proceed to the Expanded Discussion for Chapter 2, where we examine these concepts in more depth, including potential tuning systems beyond the standard 12-TET.

Chapter 2: Mathematical and Musical Foundations

Expanded Discussion

February 9, 2025

2.1 Historical and Theoretical Context

Early Observations The relationship between music and mathematics goes back to the Pythagoreans, who famously studied how simple integer ratios (e.g., 2:1, 3:2) produced consonant intervals like the octave and the perfect fifth. Over the centuries, theorists recognized that pitch perception is **logarithmic** in frequency: a doubling of frequency corresponds to a consistent musical distance (an octave).

Euler's Tonnetz and Beyond

- **Leonhard Euler (18th c.)** introduced concepts for visualizing harmonic relationships in a 2D lattice called the *Tonnetz*, where chords appear as adjacent triangles or parallelograms. This lattice concept shaped later approaches to pitch geometry.
- **Neo-Riemannian Theory (19th–20th c.)** refined these lattices, focusing on transformations (P, L, and R) that move triads within the lattice. While we won't delve deeply into chord transformations here, the broader lesson is that *geometric representations* of pitch or chord space can be powerful.

2.2 Psychoacoustics of Pitch

Log-Frequency Perception

Human pitch perception scales roughly with log(frequency), meaning:

$$\Delta(\text{pitch}) \approx \Delta(\log(\text{freq})).$$

For example, going from 220 Hz to 440 Hz (octave) feels like the same distance as 440 Hz to 880 Hz. This is why *octave equivalence* is so compelling: doubling frequency is heard as "the same note, higher."

Octave Equivalence vs. Height

- **Equivalence:** Listeners perceive G_3 and G_4 as "the same pitch class," even though G_4 is higher.
- **Height:** We also hear G_4 as higher than G_3. Thus, pitch has a dual nature: *circular* (modulo octave) and *linear* (ascending frequency).

A **spiral** or **helix** unifies these aspects.

2.3 Scales and Tuning Systems

Equal Temperament vs. Just Intonation

- **12-Tone Equal Temperament (12-TET)** divides the octave (2:1 ratio) into 12 equal steps (semitones). Each semitone multiplies frequency by $2^{1/12}$.
- **Just Intonation** uses integer ratios (e.g., 3:2, 5:4), resulting in pure-sounding intervals, but each key may have different tuning quirks.

For simplicity, most modern Western instruments use 12-TET. **However**, the helical model applies to *any* scheme where you define how frequencies wrap each octave.

Choosing a Scale for Sonification

- **Pentatonic or Major/Minor Scales** are less dissonant for most listeners, making them suitable for data sonification.
- **Microtonal Scales** (24-TET or 31-TET) allow finer pitch distinctions, but can sound unfamiliar or require listener training.

2.4 Group-Theoretic Insights (Light Version)

Pitch Classes as $\mathbb{Z}_{12}$ In 12-TET, pitch classes can be labeled 0 through 11 (like C=0, C♯=1, D=2, etc.). Adding 12 yields an octave shift. So mathematically:

$$\text{pitch class} = f \mod 12,$$

where f is a real number representing semitones above some reference pitch.

Octave Equivalence as a Quotient If we treat frequency on a *continuous* scale, then identifying f with $f + 12$ reflects the cyclical nature. This can be viewed as the quotient space $\mathbb{R}/12\mathbb{Z}$. In simpler sonification terms, it means *any* function value can be snapped (mod 12) into a single octave or repeated across multiple octaves.

2.5 Pitch Helix (Deeper Look)

From Circle to Helix

Consider mapping pitch class (0–12) to a circle, but letting each increment of 12 move us *one full revolution higher*. This yields a 3D curve:

$$\begin{aligned} x(\theta) &= r\cos(\theta), \\ y(\theta) &= r\sin(\theta), \\ z(\theta) &= k\,\theta, \end{aligned}$$

where θ could be measured in semitones, or continuously if you allow microtonal intervals. Each 2π revolution in θ corresponds to an octave shift in z.

Examples of Spiral/Helix Models

- **Shepard's Pitch Helix (1964):** Roger Shepard used a helix to depict how pitch classes repeat every 12 semitones while ascending in pitch height. He also popularized the "Shepard tones" illusions, which exploit the cyclical aspect of pitch.
- **Spiral Array (Chew, 2001):** Elaine Chew designed a spiral structure for chords and keys, embedding more complex tonal relationships in a continuous 3D space.

2.6 Why a Helix Helps Sonification

A helical (or spiral) coordinate system for pitch is uniquely suited to *musical* sonification because it encodes:

1. **Octave Equivalence**: The *circular* dimension ensures that notes differing by 12 semitones align vertically.
2. **Ascending Frequency**: The *vertical* dimension (e.g., z-axis) ensures you can hear higher octaves as truly higher.
3. **Visual Congruence**: If you plot the helix, you see a coil. If you *listen* to the helix, you hear repeated pitch classes in ascending registers.

This synergy is crucial for creating consistent, *musically recognizable* patterns in data-based sonification.

2.7 Example Use-Cases

- **Function Grapher**: When listening to $y = f(x)$, use $x \mapsto$ time and $f(x) \mapsto \theta$, so that changes in $f(x)$ shift around the pitch-class circle. Over many cycles, the function might traverse multiple octaves (vertical dimension).

- **Data Streams**: In real-time data (like stock prices), the difference between consecutive data points can be mapped to $\Delta\theta$ around the circle, so data fluctuations produce melodic arcs around the helix, with upward/downward movement in frequency space.

2.8 Concluding Remarks

In this expanded exploration of Chapter 2, we see that:

1. **Log-frequency** mapping underpins the musical sense of pitch distance.
2. **Cyclicity** of octaves (mod 12 or another step size) merges with linear ascent to form a spiral or helix.
3. **Group theory** can formalize how pitch classes repeat, but a deep dive into abstract algebra is optional for practical sonification.

From a historical standpoint, we inherit the concept of representing music in a geometric space from Euler's Tonnetz, Neo-Riemannian lattices, and advanced spiral arrays. By adapting these ideas, we create a robust framework for mapping *any* numeric data to a structured pitch space.

In the next section of Chapter 2 (Python Code Examples), we will provide short demonstrations showing how to compute and visualize basic spiral or helical mappings, preparing for the more detailed sonification steps ahead.

Chapter 2: Mathematical and Musical Foundations
Python Code Examples

February 9, 2025

Overview

This section provides Python code snippets illustrating:

- How to **plot a 3D helix** (to visualize cyclical pitch classes plus linear ascent).
- Converting **semitone offsets** to frequencies via a logarithmic formula.
- (Optional) **Playback of a simple semitone scale**, showing how these frequencies sound when played in discrete steps.

You will find two types of code below:

1. Non-interactive Python scripts (plain text) you can run from a terminal or as `.py` files.
2. A simple interactive snippet (using Jupyter, if desired).

Important: For audio playback, ensure you have the `sounddevice` package installed as discussed in Chapter 1. For plotting, install `matplotlib`.

Below we reference each code block in triple-backtick format. Copy them directly into a Python file or a Jupyter notebook as desired.

2.1 Plotting a 3D Helix

File Suggestion: `plot_helix.py`
Purpose: Visualize a pitch helix in 3D using `matplotlib`.

Code

See "`plot_helix.py`" below:
(Plain-Text Code Block in triple backticks)

2.2 Converting Semitones to Frequency

File Suggestion: `semitone_frequency.py`
Purpose: Demonstrate the formula $f = f_{\text{ref}} \cdot 2^{n/12}$ for pitch calculations.

Code

See "`semitone_frequency.py`" below:
(Plain-Text Code Block in triple backticks)

2.3 Simple Scale Playback (Optional)

File Suggestion: `scale_playback.py`
Purpose: Play ascending semitones to illustrate how the frequencies *sound*.

Code

See "`scale_playback.py`" below:
(Plain-Text Code Block in triple backticks)

2.4 Conclusion

Together, these scripts let you:

- **See** the pitch helix in 3D.
- **Compute** frequencies based on semitone offsets.
- Optionally **hear** a short ascending scale.

These examples set the stage for more sophisticated sonification techniques, which we will explore in subsequent chapters.

plot_helix.py

```python
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def plot_pitch_helix(num_semitones=36, radius=1.0, rise_per_octave=1.0):
    """
    Plot a simple 3D helix where each 2*pi turn corresponds to 12 semitones (1 octave).

    num_semitones: total semitones to plot
    radius       : radius of the helix
    rise_per_octave: vertical rise for each 12-semitone cycle
    """
    num_points = 200
    semitones = np.linspace(0, num_semitones, num_points)
    # Convert semitones to angle (2*pi per 12 semitones)
    theta = (2.0 * np.pi / 12.0) * semitones

    x = radius * np.cos(theta)
    y = radius * np.sin(theta)
    z = (rise_per_octave / 12.0) * semitones

    fig = plt.figure(figsize=(7, 5))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(x, y, z, color='blue', linewidth=2.0)
    ax.scatter(x, y, z, color='blue', s=10)  # mark each semitone

    ax.set_title("3D Pitch Helix")
    ax.set_xlabel("X (cos)")
    ax.set_ylabel("Y (sin)")
    ax.set_zlabel("Z (Octave rise)")

    plt.show()

if __name__ == "__main__":
    plot_pitch_helix(num_semitones=36, radius=1.0, rise_per_octave=3.0)
```

```
#!/usr/bin/env python

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def plot_pitch_helix(num_semitones=36, radius=1.0,
rise_per_octave=1.0):
    """
    Plot a simple 3D helix where each 2*pi turn corresponds to 12
semitones (1 octave).

    num_semitones: total semitones to plot
    radius       : radius of the helix
    rise_per_octave: vertical rise for each 12-semitone cycle
    """
    num_points = 200
    semitones = np.linspace(0, num_semitones, num_points)
    # Convert semitones to angle (2*pi per 12 semitones)
    theta = (2.0 * np.pi / 12.0) * semitones

    x = radius * np.cos(theta)
    y = radius * np.sin(theta)
    z = (rise_per_octave / 12.0) * semitones

    fig = plt.figure(figsize=(7, 5))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot(x, y, z, color='blue', linewidth=2.0)
    ax.scatter(x, y, z, color='blue', s=10)  # mark each semitone
```

```
    ax.set_title("3D Pitch Helix")
    ax.set_xlabel("X (cos)")
    ax.set_ylabel("Y (sin)")
    ax.set_zlabel("Z (Octave rise)")

    plt.show()

if __name__ == "__main__":
    plot_pitch_helix(num_semitones=36, radius=1.0,
rise_per_octave=3.0)
```

scale_playback.py

```python
#!/usr/bin/env python

import numpy as np
import sounddevice as sd

def play_semitone_scale(base_freq=440.0, steps=12, duration=0.4, samplerate=44100):
    """
    Plays a scale from 'base_freq' upwards by 'steps' semitones,
    each note lasting 'duration' seconds.
    """
    for offset in range(steps + 1):
        freq = base_freq * (2.0 ** (offset / 12.0))
        # Generate a sine wave for one note
        t = np.linspace(0, duration, int(samplerate * duration), endpoint=False)
        wave = 0.3 * np.sin(2 * np.pi * freq * t)
        sd.play(wave, samplerate=samplerate)
        sd.wait()  # wait for the note to finish

if __name__ == "__main__":
    # Ascend one octave from A4 to A5
    play_semitone_scale(base_freq=440.0, steps=12, duration=0.3)
```

```python
#!/usr/bin/env python

import numpy as np
import sounddevice as sd

def play_semitone_scale(base_freq=440.0, steps=12, duration=0.4, samplerate=44100):
    """
    Plays a scale from 'base_freq' upwards by 'steps' semitones,
    each note lasting 'duration' seconds.
    """
    for offset in range(steps + 1):
        freq = base_freq * (2.0 ** (offset / 12.0))
        # Generate a sine wave for one note
        t = np.linspace(0, duration, int(samplerate * duration), endpoint=False)
        wave = 0.3 * np.sin(2 * np.pi * freq * t)
        sd.play(wave, samplerate=samplerate)
        sd.wait()  # wait for the note to finish

if __name__ == "__main__":
    # Ascend one octave from A4 to A5
    play_semitone_scale(base_freq=440.0, steps=12, duration=0.3)
```

semitone_frequency.py

```python
#!/usr/bin/env python

import numpy as np

def semitones_to_frequency(base_freq=440.0, semitone_offset=0):
    """
    Converts a semitone offset (relative to base_freq) into a frequency in Hz.
    Example: semitones_to_frequency(440, 12) -> 880.0
    """
    return base_freq * (2.0 ** (semitone_offset / 12.0))

if __name__ == "__main__":
    base_freq = 440.0  # A4
    for offset in range(13):  # 0 to 12
        freq = semitones_to_frequency(base_freq, offset)
        print(f"{offset:2d} semitones above {base_freq} Hz = {freq:.2f} Hz")
```

```python
#!/usr/bin/env python

import numpy as np

def semitones_to_frequency(base_freq=440.0, semitone_offset=0):
    """
    Converts a semitone offset (relative to base_freq) into a
frequency in Hz.
    Example: semitones_to_frequency(440, 12) -> 880.0
    """
    return base_freq * (2.0 ** (semitone_offset / 12.0))

if __name__ == "__main__":
    base_freq = 440.0  # A4
    for offset in range(13):  # 0 to 12
        freq = semitones_to_frequency(base_freq, offset)
        print(f"{offset:2d} semitones above {base_freq} Hz =
{freq:.2f} Hz")
```

Chapter 3: Implementing the Helical Pitch Axis
Preliminary Concepts

February 9, 2025

3.1 Introduction

In Chapters 1 and 2, we established the motivation for sonification via a **helical pitch model** and reviewed the basic math/music concepts (log-frequency perception, discrete vs. continuous pitch, etc.). Now, we delve into the explicit *implementation* of a **Helical Pitch Axis**.

3.2 Coordinate Definition

Recap The helix equation in 3D can be written as:

$$x(\theta) = r\cos(\theta),$$
$$y(\theta) = r\sin(\theta),$$
$$z(\theta) = k\,\theta,$$

where θ might represent semitone steps (discrete) or a continuous parameter for pitch. Each 2π increase in θ corresponds to an octave shift in many sonification schemes.

3.3 Discrete vs. Continuous Steps

Discrete Semitones

- **12-TET:** Typically, θ advances in increments of $\frac{2\pi}{12}$ per semitone.
- **Scale Subsets:** If you want to use a major or minor scale, you map only certain θ values within each octave (e.g., 0, 2, 4, 5, 7, 9, 11).

Continuous Pitch Sweeps

- **Microtonal or Glissando:** If θ changes smoothly, you can create sliding pitches or microtonal intervals.
- **Risk of Siren-Like Audio:** Without careful structuring (rhythm, timbre, envelopes), continuous sweeps can sound more like test signals than music.

3.4 Avoiding "Siren" Sounds

Quantization and Envelopes To maintain a *musical* flavor, we often:

1. **Quantize** pitch to discrete notes in a scale.
2. Apply **rhythmic subdivisions** or short **envelopes** so each pitch has a clear attack/decay rather than a continuous sweep.

Time vs. Parameter Remember that θ might be driven by an external parameter (e.g., x-axis data). To avoid abrupt or *too-frequent* pitch changes, you can:

- Downsample or smooth the data,
- or only trigger note changes at specific time intervals.

3.5 Conclusion

Having laid out the basic coordinate definitions (the helix equations) and the pitfalls of continuous pitch, we're ready to implement the **Helical Pitch Axis** in code. In the *Expanded Discussion*, we'll delve deeper into how to map real data onto θ, choose t (time) increments, and incorporate scale quantization or microtonal approaches.

Next: The Expanded Discussion for Chapter 3, where we explore practical data mappings, scaling factors, and more nuanced "helix management" for musically coherent sonification.

Chapter 3: Implementing the Helical Pitch Axis
Expanded Discussion

February 9, 2025

3.1 Mapping Data to θ

Recap The core idea is to treat θ as your *pitch parameter*. For instance, if you have a function $y = f(x)$, you might set:

$$\theta = \alpha \cdot f(x),$$

where α is a scaling factor controlling how steeply pitch changes in response to $f(x)$.

Choosing a Scaling Factor

- **Perceptual Range:** If $f(x)$ varies from 0 to 10, and you want at most two octaves of range, you need α such that θ runs at most $2\pi \times 2$ (2 octaves) across that domain.
- **User Control:** Provide a slider or parameter that adjusts α so the user can tune how sensitive pitch is to changes in $f(x)$.

3.2 Managing r (Radius) and k (Vertical Rise)

Radius r Though r is mostly a visual artifact (the circle radius in 2D), it can also represent how "wide" the pitch classes are spaced if you do any geometric transformations in 3D.

Vertical Rise k Each 2π in θ yields an increase of $2\pi k$ in the z direction:

$$z = k\,\theta.$$

If you want every 12 semitones to shift z by 1.0, then let

$$k = \frac{1}{(2\pi/12)} = \frac{12}{2\pi}.$$

That way, each semitone yields a vertical shift of 1/12, and each octave shift (12 semitones) raises z by 1.0.

3.3 Discrete Pitch Mapping

Scale Quantization

Instead of letting θ vary continuously:

1. Compute a real value $\theta_{\text{real}} = \alpha \cdot f(x)$.
2. Convert to **nearest semitone** or **nearest scale degree** within the circle. For example:

$$\text{RoundToScale}(\theta_{\text{real}}) = \arg\min_{s \in \text{ScaleSet}} |\theta_{\text{real}} - s|$$

   where ScaleSet might be $\{0, 2, 4, 5, 7, 9, 11\}$ for a major scale (in semitone steps).

Implementation Detail If θ_{real} is measured in semitones, you can do $\text{round}(\theta_{\text{real}})$ and then mod 12 (plus a scale subset). Alternatively, keep a `lookup table` or a direct `if` test to find the closest note in your scale.

3.4 Time vs. Parameter Revisited

Time Axis The simplest approach is to increment a time index t in small steps (e.g., each 8th note or 16th note) and compute $\theta(t)$. If $f(x)$ is a function of x, you can either:

- Step x linearly in time, i.e. $x(t) = x_0 + \Delta x \cdot t$,
- or let x come from real data (like stock prices or sensor readings).

Musical Subdivisions Use a **tempo** (beats per minute) to define how often you sample θ. This ensures that changes in pitch align with a rhythmic grid, making them more recognizable as melodic/harmonic gestures.

3.5 Practical Tips to Avoid Chaos

- **Smoothing the Data:** If $f(x)$ is noisy, a pitch jump on every tiny fluctuation can be distracting. Applying a rolling average can keep the melodic contour more pleasing.
- **Limiting Range:** Constrain pitch to a comfortable range (e.g. $\theta_{\min} = 0$, $\theta_{\max} = 24$ semitones above some base). This prevents extremely low or high pitches that might be inaudible or unpleasant.
- **Dynamic Variation:** Consider applying an amplitude envelope or velocity changes to reflect magnitude or derivatives of the data, so the sonification has expressive contrast.

3.6 Conclusion

The **Helical Pitch Axis** merges cyclical pitch-class recognition with linear ascent across octaves. By carefully mapping data to θ, discretizing pitch to a musical scale, and structuring time in rhythmic steps, we *avoid* the classic pitfalls (continuous sirens) and *reap* the benefits of a coherent pitch framework.

Next, in the Code Examples, we'll implement a full helical sonification *function that maps data into θ, manages scale quantization, and plays short notes in a rhythmic sequence.*

Chapter 3: Implementing the Helical Pitch Axis
Python Code Examples

February 9, 2025

Overview

These code examples illustrate:

1. Converting an array of data into **helical** coordinates (θ).
2. **Scale-quantizing** θ so we produce discrete pitches rather than a continuous sweep.
3. Generating a short musical sequence at a defined tempo, demonstrating how to embed data in a rhythmic grid.

3.1 Helical Conversion and Scale Quantization

File Suggestion: `helix_quantization.py`
Purpose: Shows how to map numeric data to θ, then snap θ to a scale set (e.g., major scale in semitones).

Code Listing

(Plain-text block in triple backticks, see below.)

3.2 Rhythmic Playback Example

File Suggestion: `helical_playback.py`
Purpose: Takes the quantized pitches and plays them with a simple beat or subdivision, ensuring each data point is turned into a short note rather than a continuous sweep.

Code Listing

(Plain-text block in triple backticks, see below.)

3.3 Conclusion

Using these scripts, you can:

1. Demonstrate mapping *any* numeric array to a helical pitch space.
2. Enforce musical scales to avoid siren-like glissandi.
3. Place notes in a rhythmic context, giving the sonification a musical feel.

This completes the basic Helical Pitch Axis implementation. In subsequent chapters, we will add a **timbre dimension**, discuss polyrhythms (Chapter 6), and explore more advanced data sets.

Chapter 3: Helical Pitch Axis (Idiot-Proof Notebook)

February 9, 2025

Overview

This notebook combines all Chapter 3 code into a *single file*, so beginners do not need to manage multiple `.py` files or worry about variable scope. It covers:

- Mapping data to a helical pitch axis.
- Quantizing data to a musical scale.
- Writing the results to a CSV file (like one program).
- Reading from that CSV and doing playback (like a second program).
- An optional interactive slider demo with `ipywidgets`.

Below is the complete notebook content in triple-backticks. Simply save it as `Chapter3_Sonification_Note`
and open in Jupyter.

helical_playback.py

```python
#!/usr/bin/env python

import numpy as np
import sounddevice as sd
import time

from helix_quantization import quantize_data, semitones_per_octave

def semitone_to_frequency(base_freq, semitone_offset):
    return base_freq * (2.0 ** (semitone_offset / 12.0))

def play_helical_data(data_array, alpha=2.5, base_freq=220.0,
                      scale_set=[0,2,4,5,7,9,11], note_duration=0.4,
                      samplerate=44100):
    """
    1) Quantize the data into semitone values (helix-based).
    2) Convert each semitone to frequency.
    3) Play each note for note_duration seconds.
    """
    # Step 1: Snap data to scale
    snapped = quantize_data(data_array, alpha=alpha, scale_set=scale_set)

    for semitone_val in snapped:
        freq = semitone_to_frequency(base_freq, semitone_val)
        t = np.linspace(0, note_duration, int(samplerate*note_duration), endpoint=False)
        wave = 0.3 * np.sin(2.0 * np.pi * freq * t)
        sd.play(wave, samplerate=samplerate)
        sd.wait()

if __name__ == "__main__":
    # Example: create some data that oscillates a bit
    x_vals = np.linspace(0, 2*np.pi, 20)
    data_vals = 5.0 + 4.0 * np.sin(x_vals)  # range roughly 1..9

    play_helical_data(data_vals, alpha=3.0, base_freq=220.0,
                      scale_set=[0,2,4,5,7,9,11], note_duration=0.3)
```

```
#!/usr/bin/env python

import numpy as np
import sounddevice as sd
import time

from helix_quantization import quantize_data, semitones_per_octave

def semitone_to_frequency(base_freq, semitone_offset):
    return base_freq * (2.0 ** (semitone_offset / 12.0))

def play_helical_data(data_array, alpha=2.5, base_freq=220.0,
                      scale_set=[0,2,4,5,7,9,11], note_duration=0.4,
                      samplerate=44100):
    """
    1) Quantize the data into semitone values (helix-based).
    2) Convert each semitone to frequency.
    3) Play each note for note_duration seconds.
    """
    # Step 1: Snap data to scale
    snapped = quantize_data(data_array, alpha=alpha,
scale_set=scale_set)

    for semitone_val in snapped:
        freq = semitone_to_frequency(base_freq, semitone_val)
        t = np.linspace(0, note_duration,
int(samplerate*note_duration), endpoint=False)
        wave = 0.3 * np.sin(2.0 * np.pi * freq * t)
        sd.play(wave, samplerate=samplerate)
```

```
        sd.wait()

if __name__ == "__main__":
    # Example: create some data that oscillates a bit
    x_vals = np.linspace(0, 2*np.pi, 20)
    data_vals = 5.0 + 4.0 * np.sin(x_vals)  # range roughly 1..9

    play_helical_data(data_vals, alpha=3.0, base_freq=220.0,
                      scale_set=[0,2,4,5,7,9,11], note_duration=0.3)
```

helix_quantization.py

```python
#!/usr/bin/env python

import numpy as np
import csv

def snap_to_scale(theta_value, scale_set, semitones_per_octave=12):
    """
    Snap a real-valued 'theta_value' (in semitones)
    to the nearest note in 'scale_set' (within one octave).
    """
    octave_int = int(theta_value // semitones_per_octave)
    remainder = theta_value - octave_int * semitones_per_octave

    best_note = None
    best_diff = 9999
    for note in scale_set:
        diff = abs(note - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = note

    snapped_value = octave_int * semitones_per_octave + best_note
    return snapped_value

def map_data_to_theta(data_array, alpha=1.0):
    """
    Convert data values to a 'theta' measure (in semitones),
    scaling by 'alpha'.
    """
    return alpha * data_array

def quantize_data(data_array, alpha=1.0, scale_set=[0,2,4,5,7,9,11], semitones_per_octave=12):
    """
    Map data to theta, then snap to the nearest note in the given scale.
    Returns an array of 'snapped' semitone values.
    """
    raw_theta = map_data_to_theta(data_array, alpha=alpha)
    snapped_array = []
    for val in raw_theta:
        snapped = snap_to_scale(val, scale_set, semitones_per_octave)
        snapped_array.append(snapped)
    return np.array(snapped_array)

if __name__ == "__main__":
    # Example usage
    data = np.arange(10)  # 0..9
    alpha_val = 2.6667
    scale = [0,2,4,5,7,9,11]

    # 1) Quantize
    snapped_result = quantize_data(data, alpha=alpha_val, scale_set=scale)
    print("Raw data:         ", data)
    print("Snapped semitones:", snapped_result)

    # 2) Optional: Write to CSV
    with open("snapped_output.csv", "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["RawData", "SnappedSemitones"])
        for raw_val, snapped_val in zip(data, snapped_result):
            writer.writerow([raw_val, snapped_val])

    print("Wrote snapped_output.csv with the data.")
```

```python
#!/usr/bin/env python

import numpy as np
import csv

def snap_to_scale(theta_value, scale_set, semitones_per_octave=12):
    """
    Snap a real-valued 'theta_value' (in semitones)
    to the nearest note in 'scale_set' (within one octave).
    """
    octave_int = int(theta_value // semitones_per_octave)
    remainder = theta_value - octave_int * semitones_per_octave

    best_note = None
    best_diff = 9999
    for note in scale_set:
        diff = abs(note - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = note

    snapped_value = octave_int * semitones_per_octave + best_note
    return snapped_value

def map_data_to_theta(data_array, alpha=1.0):
    """
    Convert data values to a 'theta' measure (in semitones),
    scaling by 'alpha'.
    """
```

```
    return alpha * data_array


def quantize_data(data_array, alpha=1.0, scale_set=[0,2,4,5,7,9,11],
semitones_per_octave=12):
    """
    Map data to theta, then snap to the nearest note in the given
scale.
    Returns an array of 'snapped' semitone values.
    """
    raw_theta = map_data_to_theta(data_array, alpha=alpha)
    snapped_array = []
    for val in raw_theta:
        snapped = snap_to_scale(val, scale_set,
semitones_per_octave)
        snapped_array.append(snapped)
    return np.array(snapped_array)


if __name__ == "__main__":
    # Example usage
    data = np.arange(10)  # 0..9
    alpha_val = 2.6667
    scale = [0,2,4,5,7,9,11]

    # 1) Quantize
    snapped_result = quantize_data(data, alpha=alpha_val,
scale_set=scale)
    print("Raw data:         ", data)
    print("Snapped semitones:", snapped_result)

    # 2) Optional: Write to CSV
```

```
with open("snapped_output.csv", "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["RawData", "SnappedSemitones"])
    for raw_val, snapped_val in zip(data, snapped_result):
        writer.writerow([raw_val, snapped_val])

print("Wrote snapped_output.csv with the data.")
```

Chapter3_Sonification_Notebook.ipynb

```
#%% md
# Chapter 3: Helical Pitch Axis (Idiot-Proof Notebook)

This notebook:

1. Defines functions to **map data** onto a helical pitch axis.
2. **Writes** the quantized output to a CSV (simulating one program).
3. **Reads** that CSV back in and **plays** the notes (simulating a second program).
4. Ends with an **interactive** slider demo so you can adjust parameters.

## Prerequisites
- Python environment with `numpy`, `sounddevice`, `pandas` (for CSV), and `ipywidgets` installed.
- Launch Jupyter with the correct conda environment:
  ```
  conda activate sonify_env
  jupyter notebook
  ```

## Table of Contents
1. [Setup (Imports)](#setup)
2. [Part A: Mapping & Writing CSV](#partA)
3. [Part B: Reading CSV & Playback](#partB)
4. [Part C: Interactive Slider Demo](#partC)

Let's begin!
#%%
##########################################################
# CELL 1: SETUP (IMPORTS)
# We put all imports here. Make sure you've installed them!
##########################################################

import numpy as np
import csv
import pandas as pd  # for reading CSV easily
import sounddevice as sd
import time
import ipywidgets as widgets
import matplotlib.pyplot as plt
from IPython.display import display

print("Imports done.")

#%% md
<a id="setup"></a>
## Setup Explanation

- `numpy`, `csv`, `pandas` are used for data.
- `sounddevice` is for audio playback.
- `ipywidgets` for interactive sliders.
- `matplotlib` for plots.

If you get an error (`No module named X`), ensure you installed these packages **inside** your
environment. E.g.:
```bash
conda activate sonify_env
conda install pandas matplotlib ipywidgets
conda install -c conda-forge python-sounddevice
```
Now let's define our pitch-mapping functions.
#%%
##
CELL 2: Define Helical & Quantization Functions
##

def snap_to_scale(theta_value, scale_set, semitones_per_octave=12):
    """
    Snap a real-valued 'theta_value' (in semitones)
    to the nearest note in 'scale_set' (within one octave).
    scale_set might be [0,2,4,5,7,9,11].
    """
    octave_int = int(theta_value // semitones_per_octave)
    remainder = theta_value - octave_int*semitones_per_octave

    best_note = None
    best_diff = 9999
    for note in scale_set:
```

```
        diff = abs(note - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = note

    snapped_value = octave_int * semitones_per_octave + best_note
    return snapped_value

def quantize_data(data_array, alpha=1.0, scale_set=[0,2,4,5,7,9,11], semitones_per_octave=12):
    """
    Map 'data_array' to semitones by scaling with alpha.
    Then snap each semitone value to the nearest note in 'scale_set'.
    """
    snapped_array = []
    for val in data_array:
        # multiply by alpha to get semitone-ish value
        raw_theta = alpha * val
        snapped = snap_to_scale(raw_theta, scale_set, semitones_per_octave)
        snapped_array.append(snapped)
    return np.array(snapped_array)

def semitone_to_frequency(base_freq, semitone_offset):
    """
    For a given semitone_offset from base_freq, convert to actual frequency.
    E.g., semitone_offset=12 => double the base_freq.
    """
    return base_freq * (2.0 ** (semitone_offset / 12.0))

print("Helical & quantization functions defined.")
#%% md
## Part A: Mapping & Writing CSV <a id="partA"></a>

We'll simulate data with a small array, quantize it, then **write** to a CSV. We'll treat that CSV
as the output of "Program A."
#%%
###########################################################
# CELL 3: Simulate data, quantize, and write to CSV
###########################################################

# Let's define some simple data, say 15 points from 0..5.
data = np.linspace(0, 5, 15)  # shape (15,)
print("Raw data:", data)

# We'll pick alpha=4, meaning each unit in 'data' -> 4 semitones.
# scale_set can be something like a major scale: [0,2,4,5,7,9,11]

alpha_val = 4.0
scale = [0,2,4,5,7,9,11]
snapped_semitones = quantize_data(
    data_array=data,
    alpha=alpha_val,
    scale_set=scale)

print("Snapped Semitones:", snapped_semitones)

# Now we write to CSV:
csv_filename = "quantized_output.csv"
with open(csv_filename, "w", newline="") as f:
    writer = csv.writer(f)
    writer.writerow(["RawValue", "SnappedSemitones"])
    for rv, ss in zip(data, snapped_semitones):
        writer.writerow([rv, ss])

print(f"Wrote {csv_filename} with {len(data)} rows.")
#%% md
Now we have a CSV file **`quantized_output.csv`** in the same folder as this notebook.

## Part B: Reading CSV & Playback <a id="partB"></a>

We'll simulate a second "program" that loads the CSV, interprets the snapped semitones, and plays
them with a short beep per note.
#%%
###########################################################
# CELL 4: Read CSV & Playback
###########################################################

# We'll read the CSV using pandas, extract 'SnappedSemitones',
# and play them one by one.

df = pd.read_csv("quantized_output.csv")
print(df.head())
```

```
snapped_array = df["SnappedSemitones"].to_numpy()

def play_notes_from_array(
    semitone_array,
    base_freq=220.0,
    note_duration=0.4,
    samplerate=44100
):
    for semitone_val in semitone_array:
        freq = semitone_to_frequency(base_freq, semitone_val)
        t = np.linspace(0, note_duration, int(samplerate*note_duration), endpoint=False)
        wave = 0.3 * np.sin(2.0 * np.pi * freq * t)
        sd.play(wave, samplerate=samplerate)
        sd.wait()  # wait for note to finish

# Let's do it:
print("Playing notes from CSV...")
play_notes_from_array(snapped_array, base_freq=220.0, note_duration=0.3)
print("Done!")
#%% md
## Part C: Interactive Slider Demo <a id="partC"></a>

To illustrate the **interactive** approach, we'll:
1. Generate new data (like a sine wave),
2. Use a slider to adjust `alpha_val`,
3. Quantize & play the data, and show a quick plot.

_Ensure you have installed `ipywidgets`:_
```bash
conda install ipywidgets
```
#%%
##
CELL 5: Interactive Demo
##

@widgets.interact(alpha_val=(0.5, 10.0, 0.5), base_freq=(110.0, 440.0, 10.0), note_dur=(0.1, 1.0,
0.1))
def interactive_helical(alpha_val=4.0, base_freq=220.0, note_dur=0.3):
    # 1) Generate data, e.g., 20 points on a sine
    x_vals = np.linspace(0, 2*np.pi, 20)
    data_vals = 5 + 4*np.sin(x_vals) # roughly 1..9

    # 2) Quantize
    scale = [0, 2, 4, 5, 7, 9, 11]
    snapped_vals = quantize_data(data_vals, alpha=alpha_val, scale_set=scale)

    # 3) Playback
    for semitone_val in snapped_vals:
        freq = semitone_to_frequency(base_freq, semitone_val)
        t = np.linspace(0, note_dur, int(44100*note_dur), endpoint=False)
        wave = 0.3*np.sin(2.0*np.pi*freq*t)
        sd.play(wave, samplerate=44100)
        sd.wait()

    # 4) Optional Plot
    plt.figure(figsize=(6,3))
    plt.plot(data_vals, 'bo-', label='Raw Data')
    plt.plot(snapped_vals, 'ro-', label='Snapped Semitones')
    plt.title(f"alpha={alpha_val}, base_freq={base_freq} Hz")
    plt.legend()
    plt.show()

    print("Done playing with alpha=", alpha_val, " base_freq=", base_freq, " note_dur=", note_dur)
#%% md
Usage
Click and drag the sliders (`alpha_val`, `base_freq`, `note_dur`) to see how the playback changes
in real time. The code:
- **Generates** a small sine-based data array.
- **Quantizes** with your chosen `alpha_val`.
- **Plays** short notes for each data point.
- **Plots** raw vs. snapped values.

End of Notebook
You have now run a complete example in **one** file. No multiple `.py` files needed!
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "9dc2724b",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 3: Helical Pitch Axis (Idiot-Proof Notebook)\n",
    "\n",
    "This notebook:\n",
    "\n",
    "1. Defines functions to **map data** onto a helical pitch axis.\n",
    "2. **Writes** the quantized output to a CSV (simulating one program).\n",
    "3. **Reads** that CSV back in and **plays** the notes (simulating a second program).\n",
    "4. Ends with an **interactive** slider demo so you can adjust parameters.\n",
    "\n",
    "## Prerequisites\n",
    "- Python environment with `numpy`, `sounddevice`, `pandas` (for CSV), and `ipywidgets` installed.\n",
    "- Launch Jupyter with the correct conda environment:  \n",
    "  ```\n",
    "  conda activate sonify_env\n",
```

```
    "  jupyter notebook\n",
    "  ```\n",
    "\n",
    "## Table of Contents\n",
    "1. [Setup (Imports)](#setup)\n",
    "2. [Part A: Mapping & Writing CSV](#partA)\n",
    "3. [Part B: Reading CSV & Playback](#partB)\n",
    "4. [Part C: Interactive Slider Demo](#partC)\n",
    "\n",
    "Let's begin!"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "2195fb3b",
   "metadata": {
    "collapsed": false,
    "jupyter": {
     "outputs_hidden": false
    },
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
```

```
    "text": [
     "Imports done.\n"
    ]
   }
  ],
  "source": [
   "##########################################################\n",
   "# CELL 1: SETUP (IMPORTS)\n",
   "# We put all imports here. Make sure you've installed them!\n",
   "##########################################################\n",
   "\n",
   "import numpy as np\n",
   "import csv\n",
   "import pandas as pd  # for reading CSV easily\n",
   "import sounddevice as sd\n",
   "import time\n",
   "import ipywidgets as widgets\n",
   "import matplotlib.pyplot as plt\n",
   "from IPython.display import display\n",
   "\n",
   "print(\"Imports done.\")\n"
  ]
 },
 {
  "cell_type": "markdown",
  "id": "3ea050ee",
  "metadata": {
   "pycharm": {
    "name": "#%% md\n"
```

````
    }
   },
   "source": [
    "<a id=\"setup\">&lt;/a&gt;\n",
    "## Setup Explanation\n",
    "\n",
    "- `numpy`, `csv`, `pandas` are used for data.\n",
    "- `sounddevice` is for audio playback.\n",
    "- `ipywidgets` for interactive sliders.\n",
    "- `matplotlib` for plots.\n",
    "\n",
    "If you get an error (`No module named X`), ensure you installed these packages **inside** your environment. E.g.:\n",
    "```bash\n",
    "conda activate sonify_env\n",
    "conda install pandas matplotlib ipywidgets\n",
    "conda install -c conda-forge python-sounddevice\n",
    "```\n",
    "Now let's define our pitch-mapping functions."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "00f6db82",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
````

```
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Helical & quantization functions defined.\n"
     ]
    }
   ],
   "source": [
    "##########################################################\n",
    "# CELL 2: Define Helical & Quantization Functions\n",
    "##########################################################\n",
    "\n",
    "def snap_to_scale(theta_value, scale_set, 
semitones_per_octave=12):\n",
    "    \"\"\"\n",
    "    Snap a real-valued 'theta_value' (in semitones)\n",
    "    to the nearest note in 'scale_set' (within one octave).\n",
    "    scale_set might be [0,2,4,5,7,9,11].\n",
    "    \"\"\"\n",
    "    octave_int = int(theta_value // semitones_per_octave)\n",
    "    remainder = theta_value - 
octave_int*semitones_per_octave\n",
    "\n",
    "    best_note = None\n",
    "    best_diff = 9999\n",
    "    for note in scale_set:\n",
```

```
    "        diff = abs(note - remainder)\n",
    "        if diff < best_diff:\n",
    "            best_diff = diff\n",
    "            best_note = note\n",
    "\n",
    "    snapped_value = octave_int * semitones_per_octave + best_note\n",
    "    return snapped_value\n",
    "\n",
    "def quantize_data(data_array, alpha=1.0, scale_set=[0,2,4,5,7,9,11], semitones_per_octave=12):\n",
    "    \"\"\"\n",
    "    Map 'data_array' to semitones by scaling with alpha.\n",
    "    Then snap each semitone value to the nearest note in 'scale_set'.\n",
    "    \"\"\"\n",
    "    snapped_array = []\n",
    "    for val in data_array:\n",
    "        # multiply by alpha to get semitone-ish value\n",
    "        raw_theta = alpha * val\n",
    "        snapped = snap_to_scale(raw_theta, scale_set, semitones_per_octave)\n",
    "        snapped_array.append(snapped)\n",
    "    return np.array(snapped_array)\n",
    "\n",
    "def semitone_to_frequency(base_freq, semitone_offset):\n",
    "    \"\"\"\n",
    "    For a given semitone_offset from base_freq, convert to actual frequency.\n",
    "    E.g., semitone_offset=12 => double the base_freq.\n",
```

```
    "    \"\"\"\n",
    "    return base_freq * (2.0 ** (semitone_offset / 12.0))\n",
    "\n",
    "print(\"Helical & quantization functions defined.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2b4dc1af",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Part A: Mapping & Writing CSV <a id=\"partA\">&lt;/a&gt;\n",
    "\n",
    "We'll simulate data with a small array, quantize it, then **write** to a CSV. We'll treat that CSV as the output of \"Program A.\""
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "66022180",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
```

```
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Raw data: [0.         0.35714286 0.71428571 1.07142857 1.42857143 1.78571429\n",
      " 2.14285714 2.5        2.85714286 3.21428571 3.57142857 3.92857143\n",
      " 4.28571429 4.64285714 5.        ]\n",
      "Snapped Semitones: [ 0  2  2  4  5  7  9  9 11 12 14 16 17 19 19]\n",
      "Wrote quantized_output.csv with 15 rows.\n"
     ]
    }
   ],
   "source": [
    "############################################################\n",
    "# CELL 3: Simulate data, quantize, and write to CSV\n",
    "############################################################\n",
    "\n",
    "# Let's define some simple data, say 15 points from 0..5.\n",
    "data = np.linspace(0, 5, 15)  # shape (15,)\n",
    "print(\"Raw data:\", data)\n",
    "\n",
    "# We'll pick alpha=4, meaning each unit in 'data' -> 4 semitones.\n",
```

```
    "# scale_set can be something like a major scale: [0,2,4,5,7,9,11]\n",
    "\n",
    "alpha_val = 4.0\n",
    "scale = [0,2,4,5,7,9,11]\n",
    "snapped_semitones = quantize_data(\n",
    "    data_array=data,\n",
    "    alpha=alpha_val,\n",
    "    scale_set=scale)\n",
    "\n",
    "print(\"Snapped Semitones:\", snapped_semitones)\n",
    "\n",
    "# Now we write to CSV:\n",
    "csv_filename = \"quantized_output.csv\"\n",
    "with open(csv_filename, \"w\", newline=\"\") as f:\n",
    "    writer = csv.writer(f)\n",
    "    writer.writerow([\"RawValue\", \"SnappedSemitones\"])\n",
    "    for rv, ss in zip(data, snapped_semitones):\n",
    "        writer.writerow([rv, ss])\n",
    "\n",
    "print(f\"Wrote {csv_filename} with {len(data)} rows.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "7a7334be",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
```

```
    }
   },
   "source": [
    "Now we have a CSV file **`quantized_output.csv`** in the same folder as this notebook.\n",
    "\n",
    "## Part B: Reading CSV & Playback <a id=\"partB\"></a>\n",
    "\n",
    "We'll simulate a second \"program\" that loads the CSV, interprets the snapped semitones, and plays them with a short beep per note."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "id": "601f2a0a",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "   RawValue  SnappedSemitones\n",
      "0  0.000000                 0\n",
      "1  0.357143                 2\n",
```

```
      "2  0.714286                 2\n",
      "3  1.071429                 4\n",
      "4  1.428571                 5\n",
      "Playing notes from CSV...\n",
      "Done!\n"
     ]
    }
   ],
   "source": [
    "################################################################\n",
    "# CELL 4: Read CSV & Playback\n",
    "################################################################\n",
    "\n",
    "# We'll read the CSV using pandas, extract
'SnappedSemitones',\n",
    "# and play them one by one.\n",
    "\n",
    "df = pd.read_csv(\"quantized_output.csv\")\n",
    "print(df.head())\n",
    "snapped_array = df[\"SnappedSemitones\"].to_numpy()\n",
    "\n",
    "def play_notes_from_array(\n",
    "    semitone_array,\n",
    "    base_freq=220.0,\n",
    "    note_duration=0.4,\n",
    "    samplerate=44100\n",
    "):\n",
    "    for semitone_val in semitone_array:\n",
```

```
    "        freq = semitone_to_frequency(base_freq,
semitone_val)\n",
    "        t = np.linspace(0, note_duration,
int(samplerate*note_duration), endpoint=False)\n",
    "        wave = 0.3 * np.sin(2.0 * np.pi * freq * t)\n",
    "        sd.play(wave, samplerate=samplerate)\n",
    "        sd.wait()  # wait for note to finish\n",
    "\n",
    "# Let's do it:\n",
    "print(\"Playing notes from CSV...\")\n",
    "play_notes_from_array(snapped_array, base_freq=220.0,
note_duration=0.3)\n",
    "print(\"Done!\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "70423b4c",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Part C: Interactive Slider Demo <a id=\"partC\"></a>\n",
    "\n",
    "To illustrate the **interactive** approach, we’ll:\n",
    "1. Generate new data (like a sine wave),\n",
    "2. Use a slider to adjust `alpha_val`,\n",
    "3. Quantize & play the data, and show a quick plot.\n",
```

```
    "\n",
    "_Ensure you have installed `ipywidgets`:_\n",
    "```bash\n",
    "conda install ipywidgets\n",
    "```"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "id": "61a18938",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "d150490a72304b35b9a52e9cafe14ad2",
       "version_major": 2,
       "version_minor": 0
      },
      "text/plain": [
       "interactive(children=(FloatSlider(value=4.0,
description='alpha_val', max=10.0, min=0.5, step=0.5), FloatSlide…"
      ]
     },
```

```
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "###########################################################\n",
    "# CELL 5: Interactive Demo\n",
    "###########################################################\n",
    "\n",
    "@widgets.interact(alpha_val=(0.5, 10.0, 0.5), base_freq=(110.0,
440.0, 10.0), note_dur=(0.1, 1.0, 0.1))\n",
    "def interactive_helical(alpha_val=4.0, base_freq=220.0,
note_dur=0.3):\n",
    "    # 1) Generate data, e.g., 20 points on a sine\n",
    "    x_vals = np.linspace(0, 2*np.pi, 20)\n",
    "    data_vals = 5 + 4*np.sin(x_vals)  # roughly 1..9\n",
    "\n",
    "    # 2) Quantize\n",
    "    scale = [0, 2, 4, 5, 7, 9, 11]\n",
    "    snapped_vals = quantize_data(data_vals, alpha=alpha_val,
scale_set=scale)\n",
    "\n",
    "    # 3) Playback\n",
    "    for semitone_val in snapped_vals:\n",
    "        freq = semitone_to_frequency(base_freq,
semitone_val)\n",
    "        t = np.linspace(0, note_dur, int(44100*note_dur),
endpoint=False)\n",
    "        wave = 0.3*np.sin(2.0*np.pi*freq*t)\n",
    "        sd.play(wave, samplerate=44100)\n",
```

```
"        sd.wait()\n",
"\n",
"    # 4) Optional Plot\n",
"    plt.figure(figsize=(6,3))\n",
"    plt.plot(data_vals, 'bo-', label='Raw Data')\n",
"    plt.plot(snapped_vals, 'ro-', label='Snapped
Semitones')\n",
"    plt.title(f\"alpha={alpha_val}, base_freq={base_freq}
Hz\")\n",
"    plt.legend()\n",
"    plt.show()\n",
"\n",
"    print(\"Done playing with alpha=\", alpha_val, \"
base_freq=\", base_freq, \" note_dur=\", note_dur)"
 ]
},
{
 "cell_type": "markdown",
 "id": "3420e426",
 "metadata": {
  "pycharm": {
   "name": "#%% md\n"
  }
 },
 "source": [
  "### Usage\n",
  "Click and drag the sliders (`alpha_val`, `base_freq`,
`note_dur`) to see how the playback changes in real time. The
code:\n",
  "- **Generates** a small sine-based data array.\n",
```

```
    "- **Quantizes** with your chosen `alpha_val`.\n",
    "- **Plays** short notes for each data point.\n",
    "- **Plots** raw vs. snapped values.\n",
    "\n",
    "## End of Notebook\n",
    "You have now run a complete example in **one** file. No
multiple `.py` files needed!"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Sonify Env",
   "language": "python",
   "name": "sonify_env"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.21"
  }
 },
```

```
 "nbformat": 4,
 "nbformat_minor": 5
}
```

RawValue	SnappedSemitones
0	0
0.357142857	2
0.714285714	2
1.071428571	4
1.428571429	5
1.785714286	7
2.142857143	9
2.5	9
2.857142857	11
3.214285714	12
3.571428571	14
3.928571429	16
4.285714286	17
4.642857143	19
5	19

Chapter 4: Timbre as a Second Dimension

Preliminary Concepts

February 9, 2025

4.1 Introduction

So far, our **Helical Sonification System (HSS)** has relied mainly on *pitch* to represent data. Chapter 4 introduces **timbre** (essentially the "tone color" or spectral quality of the sound) as a *second dimension*. This allows us to convey more information simultaneously and can make the sonification feel more "musical" or expressive.

4.2 Why Timbre?

Definition Timbre is the quality of a sound that distinguishes it from other sounds of the same pitch and loudness. For instance, a trumpet and a flute playing the same note have different *timbres*.

Dimensions of Timbre Timbre is inherently multi-dimensional (involving spectral content, attack transients, vibrato, etc.). However, in this chapter, we focus on **one simple timbre parameter**—often called "brightness" or *spectral centroid*—to keep things tractable.

4.3 One-Dimensional Timbre Parameter

Spectral Centroid or Filter Cutoff

We can simplify timbre to one dimension by:

- Controlling a *filter cutoff* in a synthesizer,
- Scaling the amplitude of higher harmonics,
- Or adjusting a single "brightness" knob in a digital audio engine.

In all these cases, *one real number* (call it T) can shift the perceived "color" from dark (low T) to bright (high T).

4.4 Mapping Data to Timbre

Approach We might let the same data $f(x)$ that drives *pitch* also drive *timbre*, or we might use a second data dimension $g(x)$. Either way, we define a function:

$$\text{timbreParam} = \beta \cdot d,$$

where d is the data or some function of it, and β is a user-chosen sensitivity factor.

Example If $d \in [0, 5]$, setting $\beta = 2$ means timbreParam ranges from 0 to 10. A higher value might yield more harmonics, harsher tone, or a more intense filter cutoff.

4.5 Avoiding Complexity

It's easy for timbre mappings to become *overly complex*, overwhelming the listener. At this stage, we:

- Use **one** dimension for timbre,
- Keep pitch the **primary** dimension,
- Possibly combine with amplitude or a second harmonic to represent brightness.

4.6 Conclusion

By adding a timbre axis, we greatly expand the expressive power of HSS. We can encode additional data in a clearly audible way, so that two values (pitch *and* timbre) vary in tandem. However, we should be mindful of *psychoacoustic* factors that can make complex timbres difficult to interpret if not handled carefully.

*Next: We discuss these psychoacoustic and implementation details in depth in the **Expanded Discussion** for Chapter 4.*

Chapter 4: Timbre as a Second Dimension
Expanded Discussion

February 9, 2025

4.1 Recap of Psychoacoustics for Timbre

Multi-Dimensional Nature of Timbre While a single parameter (e.g., brightness) is convenient, research (e.g., *John Grey, 1977*, or *Stephen McAdams*) shows timbre can be conceptualized in a space of at least three dimensions:

- **Spectral Centroid (Brightness)**—the "center of mass" of frequencies,
- **Attack/Decay Transients**—how quickly a sound begins or ends,
- **Spectral Flux**—how partials change over time.

For simplicity, we reduce this to a **single timbre axis**, acknowledging the complexity of real-world timbres.

4.2 Typical Implementations of a Single Timbre Axis

Method 1: Harmonic Scaling

One straightforward approach: add or remove strength in upper harmonics. If our base waveform is a sine wave, we can add a second or third harmonic whose amplitude is controlled by our timbre parameter T.

$$\text{wave}(t) = A\sin(2\pi f t) \; + \; T \cdot B\sin(2\pi \cdot 2f \cdot t).$$

Larger T = stronger second harmonic = perceived as brighter or buzzier.

Method 2: Filter Cutoff

Another approach is using a low-pass or band-pass filter. If the cutoff frequency ω_c is:

$$\omega_c = \omega_{\min} \; + \; T \cdot (\omega_{\max} - \omega_{\min}),$$

then higher T = higher cutoff = more high-frequency content passes, yielding a brighter sound.

Method 3: Sample-based or Wavetable Interpolation

In more advanced systems, we could crossfade between a dark timbre sample and a bright timbre sample. But for a simple demonstration, adding a second harmonic (Method 1) is typically enough.

4.3 Mapping Data to Timbre (Implementation Details)

Data Range If your data $g(x)$ is in the range $[0, 5]$, define a scaling factor β. Then timbreParam $= \beta \cdot g(x)$.

- If $\beta = 2$, timbreParam can go up to 10,
- meaning your harmonic amplitude or filter cutoff might range from near 0 up to a bright threshold.

Combining with Pitch We still use α for pitch (as in Chapter 3). Now we have **two data mappings**:

$$\theta = \alpha \cdot f(x), \quad T = \beta \cdot f(x).$$

(Or $f(x)$ vs. $g(x)$ if you have separate data streams.)

4.4 Practical Warnings

- **Listener Overload:** If pitch is already varying quickly, adding fast timbre changes can confuse the ear. Consider smoothing or simpler motion for timbre.
- **Too Bright or Too Loud:** High harmonic levels can create harsh, unpleasant sounds, especially over consumer speakers or headphones.
- **Musical Context:** Some scales or chord progressions depend on timbre changes for expressiveness. A timbre axis can highlight tension/resolution in a piece, but it might also overshadow pitch if overused.

4.5 Case Study: Single Data Stream for Pitch & Timbre

Suppose we have one-dimensional data (e.g., a sine wave from 0 to 5). We might do:

$$\theta = \alpha \cdot d, \quad T = \beta \cdot d.$$

Then each data point d yields a pitch class (snapped to a scale) plus a timbre parameter. Over time, as d rises or falls, *both pitch and brightness* shift in tandem, creating a distinctive sonic “shape.”

4.6 Conclusion

Timbre provides a powerful second dimension to the Helical Sonification System, enabling richer encoding of data. By carefully choosing the mapping function and limiting the parameter range, we can produce noticeable yet not overwhelming timbral shifts. Combined with pitch, this leads to a more expressive and information-rich sonification.

Next: We offer a single Jupyter Notebook that demonstrates pitch + timbre mapping, CSV output, CSV input, and an interactive slider approach.

Chapter 4: Timbre as a Second Dimension
Idiot-Proof Notebook

February 9, 2025

Overview

In this single `.ipynb` file, we illustrate how to use **timbre** as a second dimension in our sonification. We do it in a fully self-contained manner:

- We map data to **pitch** (semidones) and **timbre** (brightness).
- We write these mappings to a CSV (simulating "Program A").
- We read from CSV, **play** the sound with a simple two-part partial (fundamental + second harmonic scaled by brightness) ("Program B").
- We include an **interactive slider** demo (`ipywidgets`) so you can adjust pitch and timbre sensitivity in real time.

Below is the complete notebook text. Save it as `Chapter4_Timbre_Notebook.ipynb` and open it in Jupyter. Run each cell from top to bottom. Enjoy!

Chapter4_Timbre_Notebook.ipynb

```
#%% md
# Chapter 4: Timbre as a Second DimensionThis minimal notebook shows how to add a **timbre axis**
to our Helical Sonification System (HSS). We do it in one file so you can simply open and run
without extra steps.## Steps1. Generate a small data array.2. Map pitch (snapped to a scale) and a
timbre parameter (brightness).3. Play them with a fundamental + second harmonic for brightness.4.
Provide an interactive slider demo using `ipywidgets`.
#%%
import numpy as np
import sounddevice as sd
import ipywidgets as widgets
import matplotlib.pyplot as plt
from IPython.display import display

# Basic pitch snap function
def snap_to_scale(theta_value, scale_set, semitones_per_octave=12):
    octave_int = int(theta_value // semitones_per_octave)
    remainder = theta_value - octave_int*semitones_per_octave
    best_note = None
    best_diff = 9999
    for note in scale_set:
        diff = abs(note - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = note
    snapped_value = octave_int * semitones_per_octave + best_note
    return snapped_value

def semitone_to_frequency(base_freq, semitone_offset):
    return base_freq * (2.0 ** (semitone_offset / 12.0))

# Map data to pitch & timbre
def map_pitch_and_timbre(data_array, alpha=1.0, beta=1.0, scale_set=[0,2,4,5,7,9,11]):
    pitch_list = []
    timbre_list = []
    for val in data_array:
        raw_pitch = alpha * val
        snapped_pitch = snap_to_scale(raw_pitch, scale_set)
        # timbre = beta * val (a simple measure of brightness)
        tparam = beta * val
        pitch_list.append(snapped_pitch)
        timbre_list.append(tparam)
    return np.array(pitch_list), np.array(timbre_list)

print("Chapter 4 imports & functions are ready.")
#%% md
## Non-Interactive Example
Here we just create a data set, map pitch/timbre, and play it once.
#%%
# Generate data and play once
data = np.linspace(0, 5, 8)
alpha_val = 3.0  # pitch sensitivity
beta_val = 1.5   # timbre sensitivity
pitch_arr, timbre_arr = map_pitch_and_timbre(data, alpha=alpha_val, beta=beta_val)

print("Data:", data)
print("Pitch semitones:", pitch_arr)
print("Timbre param:", timbre_arr)

samplerate = 44100
base_freq = 220.0
note_duration = 0.4

for ps, tv in zip(pitch_arr, timbre_arr):
    freq = semitone_to_frequency(base_freq, ps)
    t = np.linspace(0, note_duration, int(samplerate*note_duration), endpoint=False)
    fundamental = 0.3 * np.sin(2.0*np.pi*freq*t)
    second_harm = 0.3 * tv * np.sin(2.0*np.pi*(2*freq)*t)
    wave = fundamental + second_harm
    sd.play(wave, samplerate=samplerate)
    sd.wait()
print("Done playing once.")
#%% md
## Interactive Slider Demo
Move the sliders to see how changing `alpha` (pitch scale) and `beta` (timbre scale) affect the
playback. Also adjust `base_freq` and `note_duration` if you want.
#%%
```

```
@widgets.interact(alpha_val=(0.5, 8.0, 0.5),
                  beta_val=(0.0, 5.0, 0.5),
                  base_freq=(110, 440, 10),
                  note_duration=(0.1, 1.0, 0.1))
def interactive_timbre(alpha_val=3.0, beta_val=1.5, base_freq=220, note_duration=0.3):
    data = np.linspace(0, 5, 8)
    pitch_arr, timbre_arr = map_pitch_and_timbre(data, alpha=alpha_val, beta=beta_val)

    samplerate = 44100
    for ps, tv in zip(pitch_arr, timbre_arr):
        freq = semitone_to_frequency(base_freq, ps)
        t = np.linspace(0, note_duration, int(samplerate*note_duration), endpoint=False)
        fundamental = 0.3 * np.sin(2.0*np.pi*freq*t)
        second_harm = 0.3 * tv * np.sin(2.0*np.pi*(2*freq)*t)
        wave = fundamental + second_harm
        sd.play(wave, samplerate=samplerate)
        sd.wait()

    # quick visualization
    import matplotlib.pyplot as plt
    plt.figure(figsize=(6,3))
    plt.plot(pitch_arr, 'ro-', label='Pitch Semitones')
    plt.plot(timbre_arr, 'bo--', label='Timbre Param')
    plt.title(f"alpha={alpha_val}, beta={beta_val}, base={base_freq}")
    plt.legend()
    plt.show()
    print("Played.")
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 4: Timbre as a Second DimensionThis minimal notebook shows how to add a **timbre axis** to our Helical Sonification System (HSS). We do it in one file so you can simply open and run without extra steps.## Steps1. Generate a small data array.2. Map pitch (snapped to a scale) and a timbre parameter (brightness).3. Play them with a fundamental + second harmonic for brightness.4. Provide an interactive slider demo using `ipywidgets`."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
   "source": [
    "import numpy as np\n",
    "import sounddevice as sd\n",
```

```
    "import ipywidgets as widgets\n",
    "import matplotlib.pyplot as plt\n",
    "from IPython.display import display\n",
    "\n",
    "# Basic pitch snap function\n",
    "def snap_to_scale(theta_value, scale_set,
semitones_per_octave=12):\n",
    "    octave_int = int(theta_value // semitones_per_octave)\n",
    "    remainder = theta_value -
octave_int*semitones_per_octave\n",
    "    best_note = None\n",
    "    best_diff = 9999\n",
    "    for note in scale_set:\n",
    "        diff = abs(note - remainder)\n",
    "        if diff < best_diff:\n",
    "            best_diff = diff\n",
    "            best_note = note\n",
    "    snapped_value = octave_int * semitones_per_octave +
best_note\n",
    "    return snapped_value\n",
    "\n",
    "def semitone_to_frequency(base_freq, semitone_offset):\n",
    "    return base_freq * (2.0 ** (semitone_offset / 12.0))\n",
    "\n",
    "# Map data to pitch & timbre\n",
    "def map_pitch_and_timbre(data_array, alpha=1.0, beta=1.0,
scale_set=[0,2,4,5,7,9,11]):\n",
    "    pitch_list = []\n",
    "    timbre_list = []\n",
    "    for val in data_array:\n",
```

```
    "        raw_pitch = alpha * val\n",
    "        snapped_pitch = snap_to_scale(raw_pitch, scale_set)\n",
    "        # timbre = beta * val (a simple measure of brightness)\n",
    "        tparam = beta * val\n",
    "        pitch_list.append(snapped_pitch)\n",
    "        timbre_list.append(tparam)\n",
    "    return np.array(pitch_list), np.array(timbre_list)\n",
    "\n",
    "print(\"Chapter 4 imports & functions are ready.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Non-Interactive Example\n",
    "Here we just create a data set, map pitch/timbre, and play it once."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
```

```
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
   "source": [
    "# Generate data and play once\n",
    "data = np.linspace(0, 5, 8)\n",
    "alpha_val = 3.0  # pitch sensitivity\n",
    "beta_val = 1.5   # timbre sensitivity\n",
    "pitch_arr, timbre_arr = map_pitch_and_timbre(data,
alpha=alpha_val, beta=beta_val)\n",
    "\n",
    "print(\"Data:\", data)\n",
    "print(\"Pitch semitones:\", pitch_arr)\n",
    "print(\"Timbre param:\", timbre_arr)\n",
    "\n",
    "samplerate = 44100\n",
    "base_freq = 220.0\n",
    "note_duration = 0.4\n",
    "\n",
    "for ps, tv in zip(pitch_arr, timbre_arr):\n",
    "    freq = semitone_to_frequency(base_freq, ps)\n",
    "    t = np.linspace(0, note_duration,
int(samplerate*note_duration), endpoint=False)\n",
    "    fundamental = 0.3 * np.sin(2.0*np.pi*freq*t)\n",
    "    second_harm = 0.3 * tv * np.sin(2.0*np.pi*(2*freq)*t)\n",
    "    wave = fundamental + second_harm\n",
    "    sd.play(wave, samplerate=samplerate)\n",
```

```
    "    sd.wait()\n",
    "print(\"Done playing once.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Interactive Slider Demo\n",
    "Move the sliders to see how changing `alpha` (pitch scale) and `beta` (timbre scale) affect the playback. Also adjust `base_freq` and `note_duration` if you want."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
   "source": [
    "@widgets.interact(alpha_val=(0.5, 8.0, 0.5), \n",
```

```
    "                   beta_val=(0.0, 5.0, 0.5),\n",
    "                   base_freq=(110, 440, 10),\n",
    "                   note_duration=(0.1, 1.0, 0.1))\n",
    "def interactive_timbre(alpha_val=3.0, beta_val=1.5, 
base_freq=220, note_duration=0.3):\n",
    "    data = np.linspace(0, 5, 8)\n",
    "    pitch_arr, timbre_arr = map_pitch_and_timbre(data, 
alpha=alpha_val, beta=beta_val)\n",
    "\n",
    "    samplerate = 44100\n",
    "    for ps, tv in zip(pitch_arr, timbre_arr):\n",
    "        freq = semitone_to_frequency(base_freq, ps)\n",
    "        t = np.linspace(0, note_duration, 
int(samplerate*note_duration), endpoint=False)\n",
    "        fundamental = 0.3 * np.sin(2.0*np.pi*freq*t)\n",
    "        second_harm = 0.3 * tv * 
np.sin(2.0*np.pi*(2*freq)*t)\n",
    "        wave = fundamental + second_harm\n",
    "        sd.play(wave, samplerate=samplerate)\n",
    "        sd.wait()\n",
    "\n",
    "    # quick visualization\n",
    "    import matplotlib.pyplot as plt\n",
    "    plt.figure(figsize=(6,3))\n",
    "    plt.plot(pitch_arr, 'ro-', label='Pitch Semitones')\n",
    "    plt.plot(timbre_arr, 'bo--', label='Timbre Param')\n",
    "    plt.title(f\"alpha={alpha_val}, beta={beta_val}, 
base={base_freq}\")\n",
    "    plt.legend()\n",
    "    plt.show()\n",
```

```
    "    print(\"Played.\")"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython"
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

Chapter 5: Practical Sonification of Mathematical Functions

Preliminary Concepts

February 9, 2025

5.1 Introduction

In previous chapters, we explored how to map data values to a **helical pitch axis** and how to incorporate a **timbre** dimension. Now, we apply these tools to **practical mathematical functions** often encountered in physics, calculus, and related fields.

5.2 Mapping Typical Math/Physics Functions

Examples Common functions to sonify include:

- Polynomial functions ($y = x^2$, $y = x^3 - x$, etc.),
- Exponential or damped sine waves ($y = e^{-x}\sin(10x)$),
- Parametric curves (Lissajous, cycloids, brachistochrone, etc.).

Key Idea We treat the independent variable as *time* (or a discrete set of steps), while the function value controls *pitch* and possibly *timbre*.

5.3 Ensuring Musical Coherence

Scale Quantization

We still **snap** pitch values to a chosen scale to avoid sirens or continuous glissandos. If the function output is large or negative, we may need to:

- Rescale or clamp the function range,
- Shift negative values up into a valid pitch range.

Rhythmic Structuring

Instead of playing every tiny increment, we place notes on a *beat grid* (e.g., 16th notes at a chosen tempo). This ensures the result feels musical.

5.4 Handling Wide or High-Frequency Ranges

- **Downsampling or Smoothing**: If a function changes extremely rapidly, we can sample fewer points or smooth the data to avoid rapid, unmusical pitch jumps.
- **Clamping Outliers**: If $f(x)$ occasionally spikes, it might jump out of hearing range. We can clamp or log-scale the data.

5.5 Conclusion

By applying these concepts, we can transform standard math/physics functions into audible, **musically coherent** sequences or textures. In the *Expanded Discussion*, we will delve deeper into specific examples (damped sine, parametric shapes) and typical pitfalls (aliasing, huge value swings, etc.).

Next: The Expanded Discussion for Chapter 5, with more detail on function selection, domain stepping, advanced scaling, and demonstration.

Chapter 5: Practical Sonification of Mathematical Functions

Expanded Discussion

February 9, 2025

5.1 Example Function: Damped Sine

A classic function for demonstration is:

$$y = e^{-x} \sin(10x),$$

which oscillates increasingly rapidly while its amplitude decays over x. This produces a visually interesting curve and can yield a distinctive *musical contour*.

Domain Selection

We might let x run from 0 to, say, 3π, giving 10–30 oscillations depending on how we space points.

Scaling

If y ends up in $[-1, 1]$, we can multiply by some factor α to map it into our pitch range. E.g., $\theta = 15 \cdot y$. We still snap to a scale (major, minor, etc.) to maintain musical intervals.

5.2 Advanced Shapes: Lissajous or Cycloids

Parametric Curves Some shapes (like $x(t)$, $y(t)$) can be turned into pitch & timbre by letting $y(t)$ feed the pitch axis, and $x(t)$ feed the timbre axis, or vice versa. For instance, a cycloid:

$$\begin{cases} x(t) = R(t - \sin t), \\ y(t) = R(1 - \cos t), \end{cases}$$

could feed $y(t)$ into pitch, $x(t)$ into timbre.

5.3 Pitfalls and Tips

- **Excessive Range**: Many math functions can blow up to large values. Consider a bounding approach or a transform (e.g., $\log(1 + |y|)$).
- **Zero Crossings**: Some functions cross zero or go negative. This is fine as long as you handle negative pitches carefully (e.g., shift or clamp them).
- **Temporal Behavior**: If $f(x)$ is extremely wiggly, sample fewer points or apply smoothing to avoid "spastic" music.

5.4 Demonstration Structure

Approach In the code examples, we:

1. Choose a function (like $y = e^{-x} \sin(10x)$).
2. Step x in discrete intervals (like 0, 0.2, 0.4, ...).
3. Scale & snap the function output for pitch.
4. Optionally map the same or a second function to timbre.
5. Write out to CSV (like "Program A"), then read and play (like "Program B") or do an all-in-one approach.

Interactive Sliders

For added clarity, we let the user adjust parameters such as:

- α: pitch scale factor,
- $x_{\max}$: the domain limit for x,
- beats or tempo for the playback speed.

5.5 Conclusion

Sonifying math functions can illuminate patterns—like how a damped sine wave's amplitude shrinks or how a cycloid's periodic arcs might map to melodic cycles. By carefully controlling *scale quantization*, *timbre parameters*, and *time structuring*, we ensure a **musical** yet scientifically grounded experience.

Next, we demonstrate a single notebook for Chapter 5 that applies these ideas to a damped sine function (or other function you choose) in a step-by-step, "idiot-proof" manner.

Chapter5_FunctionSonification.ipynb

```
#%% md
# Chapter 5: Practical Sonification of a Damped Sine

This notebook demonstrates how to sonify a typical math function, like:
\[
  y(x) = e^{-x} \sin(10x),\quad x \in [0, x_{\max}].
\]
We'll:
1. Generate an array of \(x\) values.
2. Compute \(y\) for each \(x\), then map it to pitch (and optionally timbre) using a scale.
3. Write to a CSV (Program A style), read it back (Program B style), and do a short playback.
4. Provide an interactive slider with `ipywidgets` so we can adjust domain range, pitch scale, etc.

**Make sure** you've installed: `numpy`, `sounddevice`, `pandas`, `ipywidgets`, `matplotlib` in
your environment. Enjoy!
#%%
import numpy as np
import pandas as pd
import csv
import sounddevice as sd
import ipywidgets as widgets
import matplotlib.pyplot as plt
from IPython.display import display

print("Chapter 5 imports loaded.")
#%% md
## Snap-to-scale & pitch conversion functions
We'll reuse the logic from previous chapters: a function to snap real values to a scale, plus
something to handle semitone -> frequency conversions.

#%%
def snap_to_scale(theta_value, scale_set, semitones_per_octave=12):
    octave_int = int(theta_value // semitones_per_octave)
    remainder = theta_value - octave_int*semitones_per_octave
    best_note = None
    best_diff = 9999
    for note in scale_set:
        diff = abs(note - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = note
    snapped_value = octave_int * semitones_per_octave + best_note
    return snapped_value

def semitone_to_frequency(base_freq, semitone_offset):
    return base_freq * (2.0 ** (semitone_offset / 12.0))

print("Snap-to-scale and pitch conversion functions ready.")
#%% md
## Part A: Generate Damped Sine Data, Snap to Pitch, Write CSV
We define a function:
\[
  y(x) = e^{-x} \sin(10x),\quad x \in [0, x_{\max}].
\]
We’ll sample it at `num_points`. Then for each \(y\), we multiply by `alpha` to get semitones, snap
to a scale, and store in CSV.
#%%
def damped_sine(x):
    return np.exp(-x) * np.sin(10*x)

def generate_and_write_csv(x_max=6.28, num_points=20, alpha=15.0, scale=[0,2,4,5,7,9,11],
csv_name="ch5_damped_sine.csv"):
    x_vals = np.linspace(0, x_max, num_points)
    y_vals = [damped_sine(x) for x in x_vals]

    # Snap pitch
    pitch_list = []
    for y in y_vals:
        raw_pitch = alpha * y  # scale the function's range
        snapped = snap_to_scale(raw_pitch, scale)
        pitch_list.append(snapped)

    with open(csv_name, "w", newline="") as f:
        writer = csv.writer(f)
        writer.writerow(["xValue", "yValue", "PitchSemitone"])
        for xv, yv, ps in zip(x_vals, y_vals, pitch_list):
            writer.writerow([xv, yv, ps])
```

```
    return x_vals, y_vals, pitch_list

print("Function to generate & write CSV is ready.")
#%% md
Let's do a quick run with `x_max=6.28` (~2\(\pi\)), `num_points=20`, `alpha=15`.
#%%
x_vals, y_vals, pitch_list = generate_and_write_csv(
    x_max=6.28,
    num_points=20,
    alpha=15.0,
    scale=[0,2,4,5,7,9,11],
    csv_name="ch5_damped_sine.csv"
)

print("x_vals:", x_vals)
print("y_vals:", y_vals)
print("pitch_list:", pitch_list)
print("CSV written: ch5_damped_sine.csv")
#%% md
## Part B: Read CSV, Playback
We’ll read the file we just wrote, interpret the `PitchSemitone` column, and produce short notes.
We can also plot `(x, y)` to see the function’s shape vs. the snapped pitches.
#%%
def read_and_play(csv_name="ch5_damped_sine.csv", base_freq=220.0, note_duration=0.3,
samplerate=44100):
    df = pd.read_csv(csv_name)
    print(df.head())

    x_arr = df["xValue"].to_numpy()
    y_arr = df["yValue"].to_numpy()
    pitch_arr = df["PitchSemitone"].to_numpy()

    # Plot for visual check
    plt.figure(figsize=(7,3))
    plt.plot(x_arr, y_arr, 'b.-', label="y(x)")
    plt.title("Damped Sine + Snapped Pitch")
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()
    plt.show()

    # Playback: fundamental only for simplicity
    for ps in pitch_arr:
        freq = base_freq * (2.0 ** (ps / 12.0))
        t = np.linspace(0, note_duration, int(samplerate*note_duration), endpoint=False)
        wave = 0.3 * np.sin(2.0*np.pi*freq*t)
        sd.play(wave, samplerate=samplerate)
        sd.wait()

    print("Done playing.")

print("Function to read CSV & play is ready.")
#%% md
### Let’s test the playback
#%%
read_and_play(
    csv_name="ch5_damped_sine.csv",
    base_freq=220.0,
    note_duration=0.3,
    samplerate=44100
)
#%% md
## Part C: Interactive Sliders
We’ll let you pick:
- `x_max` (domain limit)
- `num_points` (how many samples)
- `alpha` (pitch scale)
- `base_freq` and `note_duration` for playback.

Each time you change a slider, the code regenerates the function, snaps the pitches, writes a CSV,
reads it, and then plays the notes. We also plot `(x, y)`.
#%%
@widgets.interact(
    x_max=(1.0, 10.0, 0.5),
    num_points=(5, 50, 5),
    alpha=(5.0, 25.0, 1.0),
    base_freq=(110, 440, 10),
    note_duration=(0.1, 1.0, 0.1)
)
def interactive_damped_sine(
    x_max=6.28,
```

```
    num_points=20,
    alpha=15.0,
    base_freq=220.0,
    note_duration=0.3
):
    # Step 1: Generate & write CSV
    _x, _y, _p = generate_and_write_csv(
        x_max=x_max,
        num_points=num_points,
        alpha=alpha,
        scale=[0,2,4,5,7,9,11],
        csv_name="ch5_damped_sine.csv"
    )

    # Step 2: Read & playback
    read_and_play(
        csv_name="ch5_damped_sine.csv",
        base_freq=base_freq,
        note_duration=note_duration,
        samplerate=44100
    )
    print("Done interactive call.")
#%%

#%%
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "4223bf6b",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 5: Practical Sonification of a Damped Sine\n",
    "\n",
    "This notebook demonstrates how to sonify a typical math function, like:\n",
    "\\[\n",
    "  y(x) = e^{-x} \\sin(10x),\\quad x \\in [0, x_{\\max}].\n",
    "\\]\n",
    "We'll:\n",
    "1. Generate an array of \\(x\\) values.\n",
    "2. Compute \\(y\\) for each \\(x\\), then map it to pitch (and optionally timbre) using a scale.\n",
    "3. Write to a CSV (Program A style), read it back (Program B style), and do a short playback.\n",
    "4. Provide an interactive slider with `ipywidgets` so we can adjust domain range, pitch scale, etc.\n",
    "\n",
    "**Make sure** you've installed: `numpy`, `sounddevice`, `pandas`, `ipywidgets`, `matplotlib` in your environment. Enjoy!"
   ]
```

```
},
{
 "cell_type": "code",
 "execution_count": 1,
 "id": "e263f581",
 "metadata": {
  "pycharm": {
   "name": "#%%\n"
  }
 },
 "outputs": [
  {
   "name": "stdout",
   "output_type": "stream",
   "text": [
    "Chapter 5 imports loaded.\n"
   ]
  }
 ],
 "source": [
  "import numpy as np\n",
  "import pandas as pd\n",
  "import csv\n",
  "import sounddevice as sd\n",
  "import ipywidgets as widgets\n",
  "import matplotlib.pyplot as plt\n",
  "from IPython.display import display\n",
  "\n",
  "print(\"Chapter 5 imports loaded.\")"
```

```
   ]
  },
  {
   "cell_type": "markdown",
   "id": "3714e673",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Snap-to-scale & pitch conversion functions\n",
    "We'll reuse the logic from previous chapters: a function to snap real values to a scale, plus something to handle semitone -> frequency conversions.\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "aaa67d5a",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
```

```
     "output_type": "stream",
     "text": [
      "Snap-to-scale and pitch conversion functions ready.\n"
     ]
    }
   ],
   "source": [
    "def snap_to_scale(theta_value, scale_set,
semitones_per_octave=12):\n",
    "    octave_int = int(theta_value // semitones_per_octave)\n",
    "    remainder = theta_value -
octave_int*semitones_per_octave\n",
    "    best_note = None\n",
    "    best_diff = 9999\n",
    "    for note in scale_set:\n",
    "        diff = abs(note - remainder)\n",
    "        if diff < best_diff:\n",
    "            best_diff = diff\n",
    "            best_note = note\n",
    "    snapped_value = octave_int * semitones_per_octave +
best_note\n",
    "    return snapped_value\n",
    "\n",
    "def semitone_to_frequency(base_freq, semitone_offset):\n",
    "    return base_freq * (2.0 ** (semitone_offset / 12.0))\n",
    "\n",
    "print(\"Snap-to-scale and pitch conversion functions ready.\")"
   ]
  },
  {
```

```
   "cell_type": "markdown",
   "id": "af732644",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Part A: Generate Damped Sine Data, Snap to Pitch, Write CSV\n",
    "We define a function:\n",
    "\\[\n",
    "  y(x) = e^{-x} \\sin(10x),\\quad x \\in [0, x_{\\max}].\n",
    "\\]\n",
    "We’ll sample it at `num_points`. Then for each \\(y\\), we multiply by `alpha` to get semitones, snap to a scale, and store in CSV."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "1226f727",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
```

```
      "name": "stdout",
      "output_type": "stream",
      "text": [
       "Function to generate & write CSV is ready.\n"
      ]
     }
    ],
    "source": [
     "def damped_sine(x):\n",
     "    return np.exp(-x) * np.sin(10*x)\n",
     "\n",
     "def generate_and_write_csv(x_max=6.28, num_points=20,
alpha=15.0, scale=[0,2,4,5,7,9,11],
csv_name=\"ch5_damped_sine.csv\"):\n",
     "    x_vals = np.linspace(0, x_max, num_points)\n",
     "    y_vals = [damped_sine(x) for x in x_vals]\n",
     "\n",
     "    # Snap pitch\n",
     "    pitch_list = []\n",
     "    for y in y_vals:\n",
     "        raw_pitch = alpha * y  # scale the function's range\n",
     "        snapped = snap_to_scale(raw_pitch, scale)\n",
     "        pitch_list.append(snapped)\n",
     "\n",
     "    with open(csv_name, \"w\", newline=\"\") as f:\n",
     "        writer = csv.writer(f)\n",
     "        writer.writerow([\"xValue\", \"yValue\",
\"PitchSemitone\"])\n",
     "        for xv, yv, ps in zip(x_vals, y_vals, pitch_list):\n",
     "            writer.writerow([xv, yv, ps])\n",
```

```
    "\n",
    "    return x_vals, y_vals, pitch_list\n",
    "\n",
    "print(\"Function to generate & write CSV is ready.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "b1b5305f",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "Let's do a quick run with `x_max=6.28` (~2\\(\\pi\\)), `num_points=20`, `alpha=15`."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "id": "4c48da0f",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
```

```
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "x_vals: [0.         0.33052632 0.66105263 0.99157895 1.32210526 1.65263158\n",
      " 1.98315789 2.31368421 2.64421053 2.97473684 3.30526316 3.63578947\n",
      " 3.96631579 4.29684211 4.62736842 4.95789474 5.28842105 5.61894737\n",
      " 5.94947368 6.28      ]\n",
      "y_vals: [np.float64(0.0), np.float64(-0.11708033289943327), np.float64(0.16600649637336012), np.float64(-0.17492872817557137), np.float64(0.16231848210388725), np.float64(-0.13983198233098382), np.float64(0.11445946773872807), np.float64(-0.09009409934391119), np.float64(0.06864682014181073), np.float64(-0.05081705984252264), np.float64(0.036609895302648236), np.float64(-0.025671404071418834), np.float64(0.01749714344845521), np.float64(-0.01155460432566371), np.float64(0.0073491973886238285), np.float64(-0.004454590143967349), np.float64(0.0025216521729529585), np.float64(-0.0012754752392541233), np.float64(0.0005065296411470002), np.float64(-5.9663473161637896e-05)]\n",
      "pitch_list: [0, -1, 2, -3, 2, -3, 2, -1, 2, -1, 0, -1, 0, -1, 0, -1, 0, -1, 0, -1]\n",
      "CSV written: ch5_damped_sine.csv\n"
     ]
    }
   ],
   "source": [
    "x_vals, y_vals, pitch_list = generate_and_write_csv(\n",
    "    x_max=6.28,\n",
    "    num_points=20,\n",
    "    alpha=15.0,\n",
```

```
    "    scale=[0,2,4,5,7,9,11],\n",
    "    csv_name=\"ch5_damped_sine.csv\"\n",
    ")\n",
    "\n",
    "print(\"x_vals:\", x_vals)\n",
    "print(\"y_vals:\", y_vals)\n",
    "print(\"pitch_list:\", pitch_list)\n",
    "print(\"CSV written: ch5_damped_sine.csv\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "2f3717ed",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Part B: Read CSV, Playback\n",
    "We’ll read the file we just wrote, interpret the
`PitchSemitone` column, and produce short notes. We can also plot
`(x, y)` to see the function’s shape vs. the snapped pitches."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "id": "c250aad5",
```

```
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Function to read CSV & play is ready.\n"
     ]
    }
   ],
   "source": [
    "def read_and_play(csv_name=\"ch5_damped_sine.csv\",
base_freq=220.0, note_duration=0.3, samplerate=44100):\n",
    "    df = pd.read_csv(csv_name)\n",
    "    print(df.head())\n",
    "\n",
    "    x_arr = df[\"xValue\"].to_numpy()\n",
    "    y_arr = df[\"yValue\"].to_numpy()\n",
    "    pitch_arr = df[\"PitchSemitone\"].to_numpy()\n",
    "\n",
    "    # Plot for visual check\n",
    "    plt.figure(figsize=(7,3))\n",
    "    plt.plot(x_arr, y_arr, 'b.-', label=\"y(x)\")\n",
    "    plt.title(\"Damped Sine + Snapped Pitch\")\n",
    "    plt.xlabel(\"x\")\n",
```

```
    "    plt.ylabel(\"y\")\n",
    "    plt.legend()\n",
    "    plt.show()\n",
    "\n",
    "    # Playback: fundamental only for simplicity\n",
    "    for ps in pitch_arr:\n",
    "        freq = base_freq * (2.0 ** (ps / 12.0))\n",
    "        t = np.linspace(0, note_duration,
int(samplerate*note_duration), endpoint=False)\n",
    "        wave = 0.3 * np.sin(2.0*np.pi*freq*t)\n",
    "        sd.play(wave, samplerate=samplerate)\n",
    "        sd.wait()\n",
    "\n",
    "    print(\"Done playing.\")\n",
    "\n",
    "print(\"Function to read CSV & play is ready.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "b69cb1b5",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "### Let’s test the playback"
   ]
```

```
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "id": "8576613d",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "      xValue    yValue  PitchSemitone\n",
      "0  0.000000  0.000000              0\n",
      "1  0.330526 -0.117080             -1\n",
      "2  0.661053  0.166006              2\n",
      "3  0.991579 -0.174929             -3\n",
      "4  1.322105  0.162318              2\n"
     ]
    },
    {
     "data": {
      "image/png":
"iVBORw0KGgoAAAANSUhEUgAAAnoAAAE6CAYAAACBGv/9AAAAOXRFWHRTb2Z0d2FyZQB
NYXRwbG90bGliIHZlcnNpb24zLjkuMiwgaHR0cHM6Ly9tYXRwbG90bGliLm9yZy8hTgP
ZAAAACXBIWXMAAA9hAAAPYQGoP6dpAABmOklEQVR4nO3deXgT1foH8G+abhTasBTaYgu
UtWW17AXZFMuqICogCKKAchUBuV4F0StyugBsIiPBT2dy4AiKyWBYFhCJFdtkqBVq
```

gLC0t0PX8/jhO0rTpkmaSyYTv53nyTDKdzJykafv2vOe8xyCEECAiIiIij+OldQOIiIi
IyDkY6BERERF5KAZ6RERERB6KgR4RERGRh2KgR0REROShGOgRERERESgGekREREQeioE
eERERkYdioEdERETkoRjoEbmBJUuWwGAwmG/+/v4IDQ1Fjx49MHPmTFy8eFHrJrrEli1
bYDAYsGXLljKP/e233/DAAw+gTp068PPzQ0hICGJjY/HPf/7T6rju3buje/fuzmmwC2V
lZeHtt99Gq1atEBQUhMDAQDRo0ACDBw/G1q1btW1UlJcFgMGDJkiWlHqd8ppSb0Wh
ESEgIHn74YRw5csR83PTp02EwGKye03fu3DLPX1b7/vvf/1bo+URq8da6AURksXjkR
FRSE3NxcXL17EL7/8grfffhv//e9/sXLlSvTs2VVPrJrqFH374Affffz+6d++Od955B2F
hYUhJSUFCQgJWrFiB9957z3zs3zs3LlzNWypOvLz8xEXF4cDBw7gX//6F9q3bw8AOH78ONa
sWYPt27ejW7duduGrfSvb355pvo0aMHcnJykJCQgBkzZmDz5s04cOAA7rjjDowZMwa9e/e
2es7cuXMRHByMGadNoIhUw0CNyI82bN0fbtm3Njx988EE899xy9/xzuOuuuzBo0CAcP34
cISEhGrbQPbzzzjTz/9BG9vy6+xoUOH4p133rE6tmnq5uXomwLFmCxx9/HPY
uMb5t2zbs2LEDn332GR5//HHz/l69emH8+PEoKChu6ep6kep1GjRujYsSMAoGvXrqhatSp
Gjx6NJUuWYNq0aQgPD0d4eLh0rSRSRSH1O3RG6uTp06eO+993D9+nXMnz/fvD8hIQFDhw5
FvXr1UKlSJdSrVw+PPPIIzrfnTPK2nK2nhn3/+GWPHjkK2nhn3/+GWPHjkWNGjUQFBSEkSNHIisrC6mqR
8eDCqVq2KsLAwPP/888jY9h4i/fhz
Dhg1DrVq14Ofnh+joaHz88cfFjjt69Ch69e6+6NgIAABAcHY9y4cbh+/Xq53pPLy8y8jODj
YKshTeHlZ/1ormrotnFKbNWsWIiMjUaVKFcTGxmLXrl3FzpeQx4FGxFpeOkeMQeYhCaPCqpR77fr7Vo2fP+wwQH2aT
8eNy4ccPqWIPBgPHjx2ZU+T/ZU9+6vRVp2KsLAwPAS2Rlk+T+X
DHg1DrVq14OfnhH+joaHz88cfFjjt69Ch69e6+6NgIAABAcHY9y4cbh+/Xq53pPLy8y8j0Dj
YKshTeHlZ/1ormrotnFKbNWsWIiMjUaVKFcTGxmLXrl3FzpeQkID7778f1atXh7+/P2J
iYvDVV1+Vq51quXz5MgAgLCzM5tcLv2bl+x0fH49//OYzI=0fH49//OMfCA4ORo0aNTBo0CCcP3/e6nk
rV65EXFwcwsLCUKlSJURHR2PKlCnIysqyOm7UqFGoUqKDh06hHvuuQeVK1dGzZo1MX7
8eNy4ccPqWIPBgPHjx2P+/Plo3Lgx/Pz80LRpU6xYsaJYu1NTU/HUU08hPDcvr6+iIy
MxGuvvYa8vDyr486fP4/BgwcjMDAQJpMJ4Q4YMQWpqavnfQBuUoE/5eSmauq1Xrx4OHTq
ErVu3mtO+9erVM3/92rVr+Oc//4nXH06NFi1yrP54zIIWdijR6Q
Dffv2hdFoxLZt28z7kpKS0KRJEwwdoHtTVq1dHSkoK5s2b2bh3bt2uHwe4cMIDg62OseBoaGoqffvoJEyZMYZMQFp
aGl599VUAIULF9CtWzf4+PC7ty5CAkJwdKlSzF+/PhyfxR+/PhyfxbH49NMWHCBAwfPhy
tW7eGj4+PXe/pxx9/9P/dGHw4dMk
nn8BkMmHFihUYMmmQIbty44bKUXtu2beHj44OJyfi3//+N+6/SwfRQ6+sGz7FmDf0K9fPyx
btgzJycn417+/+hUcffdRQ///yz+Zjjx4+jb9++ymRpEipXroyRj4/i7bffxu7du7du6204D
c3Fz07dsXTz31FKZMmhIpdO3bg9ddfx19//YfXbtx1cf+/yZf+6WrEs8ZdjxoWzZqFy5MubOnYt
HHnkE3t7eeOihhXhwDIIK99+b8vL8VPCv+//9bzzRo0AA7d+/7E66+/jqSk9JCaevBgcPPmfTfT
s2RPnz5/HzJkz0bhxY/MX8pmqrY5/pIRMAA1a9a+VSq1bhoYcgcsalkqf//fz
8DAXr1/HXXfdxhSKkJLz44ovo0KEDjMzsW3bNqSkpcAqKsp8nvJ8zoicShCR5hYvXiw
AiD179pR6zdaaLkK+i41Vi7x6l5eSIzM1NUrxZVP/++8XO/eyz1odP3DgQAFAzjJo1ya2r
/nXfeKVq3bm1+fPr0aQFA1K5dW9y9yedXdu/o8PyPdsZfv9yv1AvBnaXQ1Vd6Yy84VvYr
z/Pjxwt/fX1y5ckUIIcSLL74oDADSoDAaDSExMtDzgo3nvvFQBEfHx4kCALXfdJQA
IAMLHx0dTRJzJW5U1y/ft3q2G7duolua3boVez0tWrQQeXl55v27d+8WAMTy5ctLfF/y8/NFRbm5ooCLHnlaiCHx79p+ZUIDe3bz53bvzzi
iokRMTIzIc21Omf//v1FWF1iYyM/PL7Wd3mxtbGiPAGC1lzc3tt8zzvzCCH
LRJUqVcyvyOSwsTIwcOVJs27bN6jj+/30009X7T8X/nnXcEAJGSkmLz/AUFBSI3N1ds3bpVAAR2N1ds3bp
VABD79+83f+2x4xxTKwI8ScYbbwgA4pdffjHvAwVkDERAgEQHiI/PFwDEypRrRW5urrhbkCYYUSXxSHB6y
+p5Sl556SlLIcfK89/df57v8p//KwCI+vXrx81cEOMGEmDvngAgvXr+++rmsuxyXSl/S1/S
WCSGEiI+PFwDEypUrRW5urrhbkCYYUSXxSHNWtm9dlRzJg
xQwAQGzduLPG69nzOiyJqVsinRBFxnVlZmbixRdfRMOGDeHt7t7Q1vb29UqVIFIKu0LdfZTsH1FN/M

VFSAEQGzduLPG69nzOiyJqVsinRBFxnVlZmbixRdfRMOGDeHt7Q1vb29UqVIFWVlZVrM

JFf3797d6HB0dDQDo169fsf1F078AMGjQIPj7+5sfBwYG4r777sO2bduQn5+PW7duYfP
mzXjggQcQEBCAvLw8861v3764deuWOWUVHx+PZs2aoVWrVlbXGDZsWLneixo1amD79u3
Ys2cP3nrrLQwYMAB//vknpk6dihYtWiAtLa3Mc/Tr1w9Go9H8uGXLlgAsqbwTJ07g6NG
jGD58OAAUez0pKSk4duxuxYqddo0KABfHx8zLfRo0cDgNU+Hx8fzJgxo8z2PvHEEzh79iy
WLVuGCRMmICIiAl9++SW6deuGd999t9jx999v9Xjoq8PAE6dOoVhw4YhNDQURqMRPj4
+5kkdtj5DynuhUL5f8fHxVvvvueceeq7GkRqMRQ4YMwYkTJ3D27FkAwNq1a9GjRw/Url3
b6r3t06cPAJhnEsfHxyMwMLDY6ynvZ0UxZMgQPj4ICAgAF27dkV+fj6++eYb8/tijx9
//BGNGzcu1+Sosj5nRM7G1C2RDmRlZeHy5cto0aKFed+/4ofy4n9zMd5zH2BFz0eVgfXhmLZBIWLD
gMBjQt29f3Lx5s9g5qlevbvXY3xP23bt0q9vzQ0FCb+3JycpCZmYnMzEzk5eXhwwW8
/xIcffmjzdSgB2OXLlxEZGVmua5Smbdu25skrubm5ePHFFzF79my88847xSZ3FF79my88847xSZlFFWjRg2
rx0paTnnvLly4AAB4/vnn8fzzz9s9R2Zo1a5CdnW1+vHbtWrz22mvYs2eSkFxcSZ8I
u9TwKk8mERx55BI888ggA4NChQ+jZsyemTZuGsWPHomrVqZjy3p9mZmZ6NKlC/z9/fH
666+jcePGCAgIQHJyMgY+Tt7V3snMr3SxlDWHR/SceGh4fjwoULWLNmTYkp98K
fFVsTkOz9rLz99tu4++uuuuu++67YTQaERwcjIiICLueX9ilS5dQp06dch1b1veByNkY6BHpwA
//ID8/HzzpIL09HSsXbsWr776KqZMNfk9NTYWvry+qVKkCHx8
fGI1GjBgxAs8884yzNUo8X0X5+PjgjehjHGcOnUqBg0
aZPOYJk2aAlHqOwoE5AHO7Cs+sdkSzZs0wdOhzQzJkzB3/++ae+ae57Ep5/Pzzzzzh//jy2bNl
iVZrl2rVrNo/p7Py8uXrQIX5ftVNJgp7XurHBscHIyWLVvijjTfesHk9jitUaMGdu/
eXeL5yqt+/9a1Y090wSuTumbonc3JkzZ/D888/D888/D8888/DZDLhqaeeAiBnNgohzL0Dik8
//RT5+flOacd3331n1dn1dN3/fp1rFmzmzBl26dIHRaERAQAB69Oy/FkH5DdhozLqDI
8jx49cOjQIezfv9/qGsuWLTStTJk3JSUmzuV9KN5e0hK02TJk3QqFEj7N++PEj7N+/3+q1tJ+qVxK+uaunSpVaPle9X0YLUmbxDmeNOLLCIrj0XwcPS4PmcutpUTh
+ODg9Y9erVw9atW7W7b1SrVg0AEBAQgiMOjlWHo+LadZfqgmDbmmHmKoNn6kxbd3KbxBYfXtsN+dK3TGsI7KcSjOTWa1EWttdhp6fbNHVwM2bhRWLEi8lBRMIjzmgnOo+bOCMfy6HFu6JjatAM+BVp89D1TOOqpZW07gPxNqsvTk3jrr7w3VDDBhdLuIPhMb5rxmNFYbAF3tbGlD9w6lVNlW
0wlsBrDZ8RR6Zf/MrUwkVyrekRMvVcrkPQzRntlfeVZFbBWTc6OWfCT9sqxkfMAbBLK+DZxL4wPUg9eccd6IpjLRhsDzVnWOHxtUJAHHiNeLDe6/K5ayLGM8BiTcuE0WVkcJvAd9gdUE8v38d/KjPqgD1kVGWgjsOCrxfFeOlUOYqHyGLC5uW/lxfg94g4cvuffYBuCmqFy29+kqI/Zp3V0UmeWvlJ7tsdxjERqY9azKnggFgNywukJtESD3oP8L3x9bvtkN4pAt+Ikp1I9SNUqAnf2zn5uK+exnPMjT5QD3+cSC3xgJTy5kp6NRyVzW6jxlNkxFXS5FX0F6hPm/ErHWxYTmMBPuhkQsxfGQAkBq5SlMPkMQJaaLZZR/tF5DXQWL3i0GWLgvFL2C5gRhlPjwTH/Ejq+hOmzKpepo8/oYVoJTABp3ZYAw8/vr4Ya5u/SZ6ss2mOZS/mRI+sg9QQVzHRyjLFnyqxjC+zikK9Zpt/uQ+UwPQ4MdCwh3L6MKYgQiLtSH8/GfYe6F4jhrQqRi8Bpmsp4hG4Qd5KG0HD5lMzZmVdlwOkQuhOaazr+HmeSlb1s/T5krkH5czEcr3Ep/nT/v0D5++Jr2/Yx4vy9

IiIjIQzHQIyIiIvJQrKOngoKCApw/fx6BgYHmavNERERERziCEwPXr11G7dm14eZXeZ8d
ATwXnz593aIFsIiIiInslJyeXuSoOAz0VKOtdJicnIygoSOPWEBERkSfLyMhAREREudb
bZqCnAiVdGxQUxECPiIiIXKI8w8U4GYOIiIjIQzHQIyIiIvJQDPSIiIiIPBTH6BEREZF
LCCGQl5eH/Px8rZvi9nx8fGA0Gh0+DwM9crmzZ4Hjx4FGjYAyZoUTEZGHyMnJQUpKCm7
cuKF1U3TBYDAgPDwcVapUceg8DPTIpRYtAp58EigoALy8gAULgNGjtW4VERE5U0FBAU6
fPg2j0YjatWvD19eXCwyUQgiBS5cu4ezZs2jUqJFDPXsM9Mhlzp61BHmA3D71FNCrl75
79thDSURUupycHBQUFCAiIgIBAQFaN0cXatasiaSkJOTm5joU6HEyBrnM8eOWIE+Rnw+
cOKFNe9SwaBFQty5w991yu2iR1i0iInJfZS3XRRZq9XjyHSeXadRIpmsLMxqBhg21aY+
jSuqhPHtW23YREREpGOiRy4SHyzF5hc2fr990pyf2UBIRkWdhoEcu9cADlvtVq+p7Ioa
n9VASEVHFdO3aFcuWLSv38Q899BmzZrlxBZZMNAjlzp+3HL/2jUgI0OzpjjM03ooiYj
IfmvXrkVqaiqGDh1a7uf8+9//xhtvvIEMF/wRZKBHLvXnn9aP//pLm3aopfDd26+u6
hJCLSi7Nngfh49xgT/cEHH+Dxxx+3a6JJy5YtUa9ePXxdutSJLZN0F+jNnTsXkZGR8Pf
3R5s2bbB9+/YSj62Scw2bbDr9+nPgqbl/Hjlk
/1nugV7j9Z88CubnatYWISE+EALKy7L/NnWtd7WDuXPueL0T52/j555+jRo0ayM7Ottr
/4IMPYuTIkUhLS8OmTZtw//33d2ZcsW+Pr6WsUn7733HoKDg5GSkmLed//992P58uU
VfwPLSVeB3sqVKzFp0iRMmzYN+/btQ5cuXdCnTx+cOXPG5vHZ2dmoWbMmpk2bhlatWpV
43qCgIKSkpFjd/P39nfUybbmtFe/SSkjRphmoKtz8/Hyjho0hEREXcuAFUqWL/7ZlnrKs
dPPOMfc+3Z2GOhx9+GPn5+Vi9erV5++nZ4j/0LLvGtr6aKF2ZWCJBHAP7sRBvTfJRGyJhUCaCXc2c1z7TaZNd9MGIZMzTtMdXaaW2/X7fSM6gbd0MOZPbyb3xfuyATdTp/EVNaR7XG8pTjEJxOtdRpjKjtWOn7ESk/PGy9C9wfIa+jGRtD37Qr3LLdmO5cPv9QNKt/xJGNyYd5ilLiY5dV9QvMQDiWjKxSpYMRvsncDiJDH3oZWMwJtGLI5yWuOqWcG57mP2Zcb3uKjM0+KnGj4g5LnmW6c/VzNn4mXLm5uHx6C3VTsRMQlH7yt0WjQb/eZSkA4/x81jhW1Y24iRIV3/MYAnnlI0KMw8rIQ77aH2+Dhm6YOnhMTHjuDHFq47FL7e9SaGb79QkqBxRUZHk4IZyqTx5PHfOp/3Z3XhhEghHBWRdHQ+lVbxijvfGsN+9N+hLtVIyQe4jEcWKlyMQvnBrZC1M/zr7DjwNGnOUb4MuBUh0D5xHHuqdbCXjXKrVnIv/NDeG5/bxfaxnUlJE3Nmt6Rj+VeERRRjuh0qgT5eWvjalsP16z92hw0BiMnfGyIUMGrdQ3ZBqjcs9EG/8oOdJ/5Wl36ulCYddZOMsyZ8Y62QLDVX7tCv+Ybur4AFOZuskOGSqelHAdsbY59DtAYZ3QoT7bWi6Awl1fz3R9hSRJ8pTa4pjtT4BdlAtVpG9fRSx6zrAu/SPYuhA3wl89p9BpNOvNChJxFmDMF47TBEZ+4eyw9YR3BCRMmmBmFljr+AV4Q3Bm4TYj8l3EdAKeEL6ie3xuQ8jR1TkBbmGs65kuKndMC1bCHf4fabcezF6Gu4JW8Yc7sL4pzCUYDXwxC0XGcO0gtH2GQZ1+oE4zBP5sPiJZi8bV83m+lRKlX3PIa9SxHdTeAiA7Vdj1A9cS2NQxTlHoE6ZMXLRbUrhhWkI+uqm8z4xEYx6OEUhvSSvMgzSCqu+hi4h1ae2iXJ5KDjb1rXt7h/5S2KdQHBYw7fmN5UhN6Kkq4lZWwm26XK6a+0xGd7GqAwm7CzbVqR9iqrD/jj0Gkq70QBiJq6TMQITs3sZhmBhAlCkv/ZJSSDH9wALDTTONOGyqh6y9zUGw4EIZe8wfWrDmXWIcvLmQ2nzXhoMxGZpsDISjs1ZJ5bGW3NAwy0AWE6jWDmYB8Znk7dxGvb8C38DRh2oRWN5xXwFDwdDlhuSmV0y38Gy6qzSdGAEZqSQUWu3aFQ9V+wWQQVDLmM/u0W6a9XozHt6FgcB7Ykyq8uQDrWzpjFtHlYX2FRAwYBqeOvv+8U9z1ZrRlX7UDQ9MMOdQ79vNjJHfbUuVK/3UKwpNKzvv5zdb2oUQWzkTbqVxbqWy0UE6UDgSdNjjaQCzhgsk/LvnVCS7PCMBvGSRZAE4zBjX2sEMeUOe/6rLK7Jf8uZWYz9NZpDb8A2QbGdyDhw0T/FNRKXsxcQbyK3rcWe9wMmDsdFQNu9m4X/fi6FwNcAFXHuPSx3wWMa2UkE4QY9F0WCDGfm8M0xmBkQ/qTY9e7yWdtwqxnMOeVzUz6qkdR5gXHX//1G+tuG7lIBG+q0PzQVO/3R9HNDdCb1Ri+Mo7zFDsW9dULTSKECEsyF0oqrMAIzQX5DUdk3pZuj9tK+8h8aepG7B8K5FSz3gRFbfkyEA4Te/w43w8Sb4Znqy4QVNEt3DWJH9uIdhcD9Ibvzm3kaHTXmkUxEpHWUGBn9ftcV4Uuyqa55Hhg7rkAwRlUUK7GRrrdrxe8JqGnXYYmMlh2r0a6XSP7NXQBn9Ao6Qe12vfN8F+7hfGZuxlcyBEhJVrTNDVlbxcAzdD8gIq2FQ/GKVoAlNHxuQIrPl6HJsrjwlIGwz6MJi2Po9yDWUv4/4bdAJ3V8f12DVnpufGsBB+GxmLL2K7XsHvQ7eyKXD/VZq8sb+zpiRGHLd70Ngp99HLSWT7Ry48nD3OJx2UZsTMhyvQWtGxO/4rT8BdFdBvDf9YJzaGPBkOsSwZDgwIPMhJrLlxMkKfzk24drp2/jSM2sFPeqWW+t7SmihVwPsNTk/Nh+YPbcZGWJIcpuTWlvJ7KnhZh+9kT+BR7lrG8pTr6oEKc+tlj8cQbtV3CszZGsaBfyWqL3fdHhSFEKwvtmPzYeqXr8icDDRoZB5p4xYeHTPLUTXZ/BrUnjmbNsfPh43vYjjC2RfJG7sO6CzVeoEzV9wzNjyeRXeDNPPnkE0ElSgoX7UNmpmNmckd8jmaehaFhNxz8JsCD+jvr+vdmgjm1VVsWBFM9oEehjUmtlaQRi73P2aFA1ld3TP+HaIBqkQnsTWHTRZ3Yfzu1cq8uqcL5mPzwI0Z/qV6IPgf3BU6F8Lh5lBaTg==

IWXx8fEqdC6AmXQV6APD000/j6aeftvm1JUuWFNsnyiiBPXv2bMyePVuNplEpcnKA06f
l/SZN5Fb5edBrj17hQA+QE0yUQK9HD61aRUREamrdujWuXr2Kt99+G02UP2BFDBgwwK5
zPvnkk2o0rVx0F+iRPp0+LVeOqFwZUIY/KgGSJ/ToAUCDBnLLmbdERJ4jSa+9EX/T1Rg
90q/C4/OUajie2KMHcOYtERG5DwZ65BJFJ2IA+p+MwUCPiMg+ZQ2nIgu13isGeuQSpQV
6Fy4AN2+6vEkOKy3Q4+8yIiILHx8fAM5f7suT5OTkAJAlXBzBMXrkErYCvWrVgCpV5Nq
DZ85YJmnoQeEaekqgV6+eTEtnZQEXLwIq1vEmItI1o9GIqlWr4uLFiwCAgIAAh1a18nQ
FBQW4dOkSAgIC4O3tWKjGQI9cwlagZzDI4OjgQdk7pqdAr3ANvapV5X0/P6BOHfm1Eyc
Y6BERFRYaGgoA5mCPSufl5YU6deo4HBAz0COny8wEzp2bt0TJ2awEupjpbeZt0bStomF
DS6DXubOrW0VE5L4MBgPCwsJQq1Yt5Obmat1Yt5Obmat0ct+fr6wsvL8dH2DHQI6dTevNq1pTp2sL
0OvO2pECvQQNg82aWWCEiKonRaHR43BmVHydjkNMpgZ6t1KxeZ96W1qMHcOYtERG5BwZ
65HS2xucp9Fo0mYEeERHpAQM9crrSAj1PTN0CTN0SEZF7YKBHTleeHr3z54HsbJc1yWF
lBXpXrsgbERGRlhjokVMJUXqgFxYGBAbK2V8C8fFyS0R
E+sZAj5zq0iUgPV3WzFN6uwozGPSXvrVVQ68wPY/TW7RIfj/uvltuFy3SuFZk3SukVEROQIBnr
kVEpvXt26gL+/7WP0NiGjpLStQq/j9M6eBZ58EigokI8LCoCnnmLPHhGRQI6cqLW2
r0FuPXlmBnl579I4ftwR5ivx8/b0OIiKyYKBHTnXsmNyWFujprZaepwZ6jRoBRYuwG42
W10NERPrDQI+cqjw9ekzduofwcODVDVV633zZ8v9xMRkT4x0COnKm1VDIWnpW6VQC81Va7
zqydFv0/9+mnTDiIiUgcDPXKawuO7ytOjd+4rpWex5ICvWrVgBo15H299eodPmz
9eM8ebdpBRETq0F2gN3fuXERGRsLf3x9t2rTB9u3TvySnNm2LVVqlVOav3t5cwZIcCcH8PMDIiJKPi4kRM7ILShw/xmeN24
AFy/K+yUFeoB+x+kdOiS33t5ym5CgXVuIiMhxugr0t28funTxgUjRhHeGuGnMbt28funTpgj5
9+uDMmTM2j8/OzkbbNmjUxbdo0tGrVyuYxXO3fuxJAhQzBixj378fI0aMwODBg/Hbb78
586XcFpS0bcOGclB/SQwGoE4ded/d07ydl1dBT6HWcntKj16eP3LJHj4hI33QV6M2aNu
jR4/GmDFjEjB0djTlz5iAiIgzZ5s2zeXy9evXxzTvvvvvvY+TIkTCZTDaDmPmTNnDu69915MnTo
VUVFRmDp1Ku655x7Mm655x7MmTPHia/k9lCeiRgKvcy8pjc7tkjVzc
hIiJ90k2gl5OTg7179yuLs5qf1xcHHbs2OGy58nOzsa1a9es/pGyBOtymH9hZXPsKx+0TCQ
qEui5+8xbTw70hLc//vCRElUw3gV5aWhry8y8jbr8/MREhJitT8T8kJAS
pqakVPm9qaqd55w5cyZm3pY30NNj6lZ2ZtKsdMtmwpH3O
cHhGRfukm0FMYDAarx0KIYvsq6yfMWSFZ6auk1OBm7dcma
L1KNMxGjaVG7btZNbjtMjItIv3QR6wcHBMBqNxRaRKaL68W9BQUF
WN7J286acdQvcnqnbmjWBwEA5vu30aWe3aWe3Sh2Fe/QAoG1buWWgR0SkX7oJ9Hx9fdGmTRts
s3LjRav/GjRvRqVOnrCpeD0Cp83Nja22Dk3bNjg0DlJpiyFkDNTa9Ys+3gldZcLMeJuavyBno
Gg6VXTy/pW6VHr1kzuVV69PbuLb4GLhER6Y031wx+TJkzFixAi0NjbVrV7z
cOYNx48YBkCnrcc8c8c+fO4fPPPzcfZeCGEvRvbXj8mxv3z
99tsYMGAAvv/+e2zatAm//+rPn5VFLLy1+fJyk8PPq88cmfWwMMDHRxnJENPn/eUm7FnZS3hp6iQQN
g3z59TMjIzbV8z5QevWbN5Fi9jAw5UaO01U2IiIMg96SrQGzJCCZCzs
3x7p161D37+6glJSUjIj/LnNj37+6glJSUYjX1YmJizPf37t2LVpYa2uVR6GR6ZXJKWfK1jQRSokP2
raNCgAVauXQ1HbnTq47HV5InsmYgCAl5cM7k6elL1m7hjolbeGnkJPM29PnJA9qVqwgtVCp
be3sDMTHAzp0yfctAj4hIf3QV6AH5AkHKN
No0LsDfQA2UumBHpduzqjVY4pb9pWoadAr/BEjMI9sO3aWQK9Rx/Vpm1ERFRxuhmjR/p
S0UAPcN8JGfYGenoqsVJ0IoaCM2+JiPSNgR45RUUCPXevpVfRHr2kJDkGzp0VnYihUAK
9ffvce5IMERHZxkCPVHf1KnDpkvzZVY+eBX/qMydhTIRsS+futfTe+ZEUb+dvRSmXXq
VMm7dsgSDRESkHwz0SHWGtGPcW2TVVKzFMUXFf/ed+f0bW6sHRtRqfbwY4ZuoR4SLLy+gT1
TRt
5n+lbIiL9YaBHqqtI2JawpG7PnAHy8QETwwqqc3r4GJPA1uzn3mZfUTiPMvyFgDLuE0l
SbV

FQI9UV9FAr3ZtWdIjNxdISVG/XY6wt4aeQg+BXuG0rZeN3whKoMc1b4mI9IeBHqlOCfT
srbvm7W3pUXK39K29NfQUelgdo+gat0UpS6H98Yd+1u0lIiKJgR6prqI9eoD7zrytSNo
WsJRY0UuPni116wLBwXJSyf79rmsXERE5joEeqUoIxwI9d515W9FAr3CPnruuF1tSaRW
FwcD0LRGRXjHQI1WdPy/HsxmNQGSk/c9315m3FQ306tSRKensbODcObVb5bi8PMuM25J
69AB9T8g4exaIj5dbIqLbDQM9UpXSm1e/PuDjY//zPS116+1teY47jtM7eRLIyQECAiz
vvS3KOD29BXqLFsnXdffdcrtokdYtIiJyLQZ6pCpH0raA56VuAfeeeauMz4uOtj3jVqH
06B05Aly/7vx2qeHsWeDJJy0p84IC4Kmn2LNHRLcXBnqkKkcDvcK19NxpTJunB3oljc9
ThIYC4eFyDOa+fc5vlxqOHy/+GcrPd8/vAxGRszDQI1Up470qGuiFh8uepexs4MIF9dr
liIrW0FO4c4mVskqrFKa3cXqNGhXvpTQaLd8PIqLbAQM9UpWjPXo+PjLYA9xnQkZFa+g
p3LnESlmlVQrT2zi98HBg4EDrff/5j+XzRUR0O2CgR6rJzQVOnZL37S2WXJi7TchwJG0
LWKduhVCjRerIzweOHpX3y0rdAvrr0QOAa9fk1miU27w8zZpCRKQJBnqkmtOnZfAQECC
XM6sod5uQ4WigFxkpa9FlZgKXLqnVKsedOiVT5JUqle+1KT16p04BV644tWmquHED+OU
Xef/FF+V26VL3CraJiJyNgR6ppnDa1mCo+HncrZaeo4Gen59laTd3St8q4/PKmnGrqFb
NkobWQ+HkX3+VpWPCw2Wg5+8vx5AmJmrdMiIi12GgR6pxdHyewtNSt4B7zry1Z3yeQk/
p202b5LZnTyAoCLjvPvl46VLt2kRE5GoM9Eg1agV6npa6Bdw70CvP+DyFXgM9ABg2TG6
XL5dDDIiIbge6C/Tmzp2LyMhI+Pv7o02bNti+fXu2/duhVt2rSBv78/6evj
q60uWLIHBYCh2u3XrljNfhkdSO9D76y/MpZDzdqcSKPaVVFHpZ8zYtVLv755
75LZPHzlr+vx5oIxfG0REHkNXgd7KlSsxadIkTJs2Dfv27UOXLl3fl3Qp08fnDlzxubxxp0
fRt++fdGlSxfs27cPL730EiZMmIBvv/3W6rigoCCkpKRY3fz9/V3xkjjWoFeRIQc43f
zpvaTFxytoadwtx69wjNu7Qn0YmLkeL5z54CUFOe0TQ3x8fKfhObNZbFnQI6VfOgheZ/
pWyK6Xegq0Js1axZGjx6z6NMWPGIDo6GnPmzEFERATmzZtn8cxAdHY0
xY8bgiSeewH//+1r4wwGA0JDQqbM/MTPnHH5Cfah3h62uZZtat1Gnonoa2Kdwv0Tp8
Gbt2SExQiI8v/vCpV5OQNwLBMGRGRp4JeTk4O9e/ea9gO/ciLi7Oan9
cXBx27Nhh8zk7d+4s1r5Xr17Ys2cPzNm5vRuBiATr6ekZ16BMz2jU8lvAaxl3v
9eoc1+1jDx6WzLGX6Rsdc1/PWc
yrG63u+PH5TY4GKhe3fzvmWzpRiZy/CvqDQdH8f1pqLJjrz0jkL4tKd
r2hW44w5ZX2/9epc3i4jI5XQT6KWlpSE/Px8hISFW+0NCQpCammrzmNTU
aGgAgKioKS5YswerMV+Vq7F8+XOOK5tQamfcxtRK2yaXLbfK6Pwanca
tWoFe5cqWFKI7jNNdu2kxYXrFxWFwR7e4TMk6dkjvzbCIYoKEUKR
AmxCi2L6yji9+8v+rOnSBV999ZUaTvPqpnTfbJhVGhGeSeEsbeL0DZfqN
yRV+Ox1ACPUdWxCjMXWbeqhXoAe6Vvq1IaRVF4aXQ3GGyTFGbN8ttx45AYGDxryvp2zV
rAHbGE5Gn002gFxwcDKPRWkzi2hFxwcDKPRWKz37uLFi8V67RShoaE2x/f29cbly5dtHmPl5yv
Pz88PQUFBFVrfnbnN69DwldQtYAj136NFzJNBr1uSvpW0VMTExC
//7msWUREmtBNofr64s2bduiW7XNBOOfr64s2bdpg48VoE4
zhBBITExEWFiYOg2/Tagd6Hlij54yTk/rHr2CAuDIEXm/pDIGbUt
0Sgr0DAZLrx7Tt0Tk6XQT6AHA5MmT8emnn6nnn+Kzzz7D4cOH46mnnkKdOHGRmyCpYSEz0TtA5kYVUi
ypPn4cePG4a+//sLvx//558LVQv+vPnM0ZOwRbIVzp33f/DwIcK4bjt5fV2o/h9c/
0EhMTzeeksgnhvEBP61p6Bp3cjH5UFMr/CMd7yLm6F+ISgqCmOa
uN20CLlxwTduIiBBcvnM2bMQEpKCpo3b44NGzagZ3ez1h5
RkVi3bh2ee+45zJ49GzOGvXsbeztyY++OADPPjjg+ZqaMkVmRzUuBbT3L6
0vxJkJ1NzmI0GCy9oVAoQ6uQ...

OnS5pnsMWLQKefFKmqL28gAULgNGjtW4VEbmaPXGHm/x6Jj07dkxu1UrbKrSupadm2hY
AqlWzBHpaTshwpLSKomlToFIl4Pp1S9pea7duWZY2K2l8XmGDB8tgadcu95ggU5azZy1
BHiC3Tz0l9xMRlYSBHjlM7fF5Cq1r6akd6AHukb51ZMatwtsbaN1a3neX9O3OnXLsYWh
o+V5baKhlHdzly53bNjUcP24J8hT5+doPBSAi98ZAjxzmrEBP65m3zgz0tOpBKjzj1pF
AD3C/cXqFy6qUUlrTyvDhcrt0qXvWBCxM+ewUZjDY3k9EpGCgRw5Tu1iywtNSt4D2JVb
++ktOMvH1dXzijLvNvC2rfp4tDzwgZx8fPQokJjqlWao5erT4PqORRZ+JqHQM9MghhVN
HTN2WTevUrZK2bdJEpl8doQR6iYlAoaWjNXH1qqWmn5KOLY+gIOC+++T9ik5McZXXX5f
bxx8Hfv4Z6NEDyMsDHntMbomIbGGgRw5JTgays2UPkVISRS1M3apPjYkYioYNZaB065b
lvFrZskWmpaOi7C85oqRvly8vPgbOXWzfDmzbJkvGzJghg7wvv5QTfBISgLfe0rqFROS
uGOiRQ5S0bcOGFa/JVhKlRy8jQ53yJvZQu4aeQgn0zpyRAbKrqTERQ+Hl5T7j9CqStlX
06SNL6Jw7J4Mpd/TGG3I7apQlkK1dG1CW5J4xQxaLJiIqioEeOcRZEzEAICAAqFlT3nd
1r54yLtBkAqpWVe+8NWvKVRuEAE6fVu+85aVmjx7gPuP0HAn0/PyAhx6S990xfZuQAPz
0k/xHasoU668NGybHGebmAiNHAjk52rSRiNwXAz1yiDMDPUC79K0z0raA9SxJV4/TU3P
GrUIJ9LRc8/bMGfk59PKSK2JUhLL27TffaNPTWpo335TbRx4pvmSdwQB88gkQHCyLWP/
nP65vHxG5NwZ65BBnFUtWKOlbV8+8dVagB2g3Ti85GcjKkuO81FqqTkndHjggx+ppYfN
muW3fXvbAVkS3bjIVevUqsH69em1z1MGDwKpV8v7UqbaPqVULmDdP3p85U/veVSJyLwz
0yCHs0bOfViVWlLRtkyalrwNrjzp1ZDo6L0+78iSOpG0VRiMwdKi8707p25kz5fbBB0v
vhX3oIdnjl58vZ+FqFXRXxNmzQHw8V/ggchYGelRht25ZetqcHeh5Yo+eqwM9NSdiKAw
GbdO3QqgT6AGW9O3q1XJpN62dOAGsWCHvT5tW9vEffSRX+zhyBHjlFee2TS2LFsle+7v
vlttFi7RuEZHnYaBHFXbypPxDGxQk00fOoFUtPU9M3ao9EUOh5YSMgwfl7OiAAKBjR8f
O1bq17O28dQv43/9UaZ5D3npLjqvs2xeIiSn7+OrVgYUL5f333gN+/dW57XMU1+4lcg0
GelRhhVfEKO+SU/bSOnWrBJpqUlK3p0+7ttCtM3r0AG1LrCi9eV26yNmzjjAYLL16S5c
6di5HnTkDfP65vF+e3jxF//6yBIsQoMoWbleWU5qnim29sr927dq027SHyVAz0qMKcPT4
PsARaV6+6bqknZ9XQU9xxhwxK8vLkH3RXEMIS6DmrR+/oUdenPJWJGI6mbRVKoLdpE3D
hgjrnrIh335UlU7p3Bzp1su+5c+YAERGyx7hoORZ3sWMH8NJLtr/2zDPA9Onar7ZC5Ck
Y6FGFuSLQCwyUKSnAdeP0lOsEBalbQ0/h5WXp1XNV+jY5GcjMlMueKaljtYSEyMBCCOD
339U9d2lyc+WKGIB6gV7DhnL2bn4+8PXX6pzTXhcuAJ9+Ku+//LL9zzeZLGPdPvpILpf
mTnbvlkWqb94EoqMthdaNRtk7XFAAvPaaDHBtre9LRPZhoEcV5opAD3B9+rbw+DxnpaR
dPSFD6c1r3Fi9GbeFaTFO77ffZGoyOBho2VK982qdvp01S44T7NBBTlKoiHvvBcaNk/e
feMJ1veFl+f13oFcv2Z7u3eUEnqQkOes2KUl+fpYtk/9gJSTIsYkffOC+S9PZwlnE5G4
Y6FGFuTrQc1WPnjMnYihcXWLFWRMxFFqM01PG591zj+wlVcvgwfJ8u3YBp06pd97yuHI
FmDtX3n/5Zcf+0Xj3XSAyUy4Ap0Ou1zxB9/yAD02jWgc2dgzRo5iSY8XAZ9ytJujzw
iJ9nExcmAd+JEeT85WcvWlw9nEZM7YqBHFXLtmmUCWjNGriLQ06pHT+2JGAo
tevTUKqtSVFiYpSdt+XJ1z12WDz6QKfZWrYB+/Rw7V5UUqwOLF8v7v7ChcCPPzrevoo6dEg
G5FeuyJ7Kdetk+0pyxx2ycPXHHwOVKsmxC1aAF98y4GxY61nEY8dC2zf7r5
tptsDAz2qEKU3LyxMjqNzJi1Tt87i6hIrrurRO30auHzZOdcoLCND9rgRgB6gd6ADB8uNw
uXeq6P9IZGcD778v7L72kzrCBbt2ASZPk/TFjgPR0x8/pqKNHgd69gfR0+T4kJQEBAXLp/u
pp2Uh7g4dgPR0uZ7vww/Lc7mLvXtluj82tvhnRQiga1f58/7MM+5To9EeETEXrn+4Cvbl
z5yIyMhL+/v5o06YNtm/fXurxW7duRZs2beDv74/69evjk08+KXbMJXb9YkK0JJDhXcXH
FxaFzX3Urv7GDTuVhxjdzLzw8nmMs3ZpHwPK6dzpzfQG
QYx9XiWRCdjLnPSjUbtuc6VsZH88PNVs4nTXBF0xKrDTxdSU4wDlPzDZM8vMYaUnDf1vwtyYr
XB6XhgvmJuWuHRq9Yrs6T8GvlDYhQDh1mW5dPjiRS/FkTgGW8OL0ggUQ6Jj5ZL2/hPpl
vBtmfXqt6w7Fhh1GgOXBF6ocCD7CMXd/FxtK7wYBD2m2T8d9gSSP4cW8F8vK7tyJGKb5ll
MZ7JJh2HkF6Eg2jO3ZLmzBQOWVGJZQwvfKyFFFNbuhyVBHDZ9Os6uRxPLVEX5BWwS53Z

PnTolAgICxMSJE8Xhw4fFwoULhY+Pj/jmm2/Mx+zYsUMYjUbx5ptviiNHjog333xTeHt
7i127dpW7Xenp6QKASE9Pd/g12vLpp0J4eQkByO2nnzrlMnZ55RXZnrFjnX+txER5reB
g519LCCFq1ZLX+/13510jN1cIb295neRk511HCCHOnJHX8fYWIjvbedd55BF5nf/8x3n
XUEycKK/11FPOu8aDD8pr/OtfzruGIivL8rlbskT98+/cafkdsmqV+ue35dQpISIi5DW
bNRPi4kXHz7l3rxBNm8pzAkKMGSNERobj5y2vrCwh5s0TonFjSxu8vYUYPly27dNPhTA
a5X6jUT6+fl2INWuEeOYZIerXtzxPuYWFCTFqlBDLlwuRlua611LUxYtCbNkixMcfy7b
GxhZvKyBE8+ZC9Okj2/zii0LMmiXE0qVCbNokxIED8jz5+fZfPzlZiJ9/du7vQ1f+LXX
267En7jAI4c7/T1jr0KEDWrdujXnKCt4AoqOjMXDgQMxUFoUs5MUXX8Tq1atx5MgR875
x48Zh//792LlzJwBgyJAhyMjIwI+FBrD07t0b1apVw/ISBuhkZ2cjOzvb/DgjIwMRERF
IT09HUHlyEnY4e1b+J1X4v0WjUfY6KYOXtTB0KLByJfDf/wL//Kdzr3XtGlCtmryfmQl
Uruy8a924YTn/lSuW6zpDo0ayx2PLFplic5affgJ695alLJSePWeYPRuYPBm4/37g+++
ddx1A9ugcOiRLoDz0kHOusWoVMGiQ/Dn76y91J3wU9cEHsretXj3ZW+6MmdFTpshepFq
15HsXHKz+NRRnzsjPdFISEBUlP+MhIeqc+9YtWUR69mwZekRGyuLSd92lzvltuXBBjhe
cO9cyNMFkkit5PPus9e/is2flz3XDhrZ/R584IX8m16+XpW9u3LB8zWCQ5X169ZI/s+3
bW8rPqEEIObb68GH5GTh82HJfzXS40SjXwA4JKX4LDbV+HBwMLFliWSXFywtYsAAYPVq
2NydHzq6/cUNui95s7be179o12+txN2oE+PvLnlejseRtaV8reszhw3IMsRDWr0dNGRk
ZMJlM5Ys7nBNrqi87O1sYjUbx3XffWe2fMGGC6Nq1q83ndOnSRUyYMMFq33fffSe8vb1
FTk6OEEKIiIgIMWvWLKtjZs2aJerUqVNiW1599VUBoNjNGT16P/9s+7+q+HjjVL2WXmBj
ZjtWrXXO9qlXl9Q4dcu51Dh+W1wkKEqKgwLnX6t1bXsvZPbSzZsnrPPigc6+zfbu8Tu3
azr3O+fPyOgaDc3tAbt4UwmSS19qyxXnXyc4WIjxcXmfePOdd59Yt2bMGCPHww867ztm
zQjRoIK/TsKEQ58455zrx8ULUrWv5LLzwgnyNajp0SIjRo4Xw87P87q1XT4g5c9TpSbx
1S/aE/etfQrRoUfz3fLVq8nu1aJF8XwsrrceooED+nGzaJMQHH8ie7y5dhKhRw/bfE+U
9rF9fiP79ZU/d7NmW3i/l5uUle5wXLRLizTdlz/rQoUL06CF7Wks7f2nXtbW/ShVL76j
eb0aj+j179vToedsbRY4aNQpPPPEEunban8I6oC0tDTk5+cjpMi/hSEhIUhNTbX5nNT
UVJvH5+XlIS0tDWFhYSUeU9I5AWDq1KmYPHmy+bHSo+cMjrRrJ/wiK9uipXfTWHkK4dow
eIHs1r12TPQTOGmcGuKaGnsJVJVacPRFDERMjP6vnz8tb7drOuY6yGkbr1nK8k7P4+8u
xcp99Jmu7OavX9fPPZS9QWJhcvsxZ/PzktTp0kD2hK1cCQ4aoe43UVDnx4uRJ2dP288/
O+xx07y5LtkycKHuE3nlHziz+4gs5HrCihJCfsVmzrGcqd+ggsxcPPCB7bdTg5yffr3v
uke0/d0729v30E7Bhg5w88/XXluLdLVrI3j4hZI+m0gM2caL8HVm4l66kiTcGg/zd07S
pvDVrJrdRUbLcTWGBgbLXMj9f/t2ZP18urVea3FzZa3jhQtm3tDT5WmzJzLR+7OMjsy0
BAXJb9FbW/ps35YSkwn9Lvbzk+Mrq1eVrzMsrfVueY06elL8vCsvPl7/ntcrC2f1xvX7
9OuLi4hAREYHHH38cjz32GO644w5ntM0mQ5G/vkKIYvvKOr7ofnvP6efnBz9HF9Ysp/B
w2e07Zox87OUlf9i0TNumpMiucKNR/jJ3hXr15MBlZ0/IcMVEDIWrSqw4eyKGonJleY2
DB2WZlQEDnHMdZ5VVsWX4cBnoff018OGHgK+vuufPywOUUSf/+pcMLp2pdWuZ9nztNTm
jtVs3mUpTw6VLMmA5dgyoU0cGeU76/9csKEiWkBkwQKb+DhyQpX5mzJDvpz1pz5wcYMU
KGeApE3AMBhnYTZ4sV+pw9j9/d9whC1w/8YT8bOzZY0nz7t4tX9+BA9bPKSiQQV9Rygo
8SiCnBHVNmsiSNeUxerQMLEtLRRfl4yNfR3nCgrw8Gay3a1c8AIuPl+1XAjY1hjMUFBQ
PXAcPdvy8hZ09Kz9H7tQ5U6HUbVpampgzZ4648847hbe3t+jdu7f4+uuvzelQZ3Cn1G1
Rzp6MIYQQL70ku4A7dHDaJcotPt6SlnGVCRPkNV94wbnXeeEFeZ0iHxunWLNGXismxnn
XKCiQaWhADpR2tscfl9d6+WXnnL+gQIg77pDX2LjROdcoLC9PDpYHhPj+e/XP/+WX8tz
BwUJkZqp/fltycixDL+6/X50hCmlpQrRsKc95xx1CnDjh+DntdeGCEAMGWNJlnTuXrx1
Xrggxc6YccqA8NyBAiPHjtXkdJUlLkxM2evWynR686y75c7dsmZzAdvOm1i0uP1uTWJw
lOVn+DXP2pA9nvx574o4KBXqF/f7772L8+PHC399fBAcHi0mTJok/zT0dPa1L59e/G

Pf/zDal90dLSYMmWKzeNfeOEFER0dbbVv3LhxomPHjubHgwcPFn369LE6pnfv3mLo0KH
lbpcrAr3kZMtYhqQkp12mXD75RLajb1/XXVMZZzZ4sHOvM3iwvE6R2N8plPGAgYHOGw9
49qzll43a45dsmTtXXq9XL+ec/8gReX4/PyFu3HDONYp67jl5zSFD1D1vfr5lBukbb6h
77rL88YcQPj7y2o7O8r1yxRI4hoYKceyYOm2siIICIRYvlj9TgBCVKwsxf77tn68TJ2Q
wFxBgCZbCwuTYs8uXXd70cktOLj52zhljwFzNFQGYKzn79bgs0Dt//rx46623ROPGjUX
lypXFyJEjxb333iu8vb2L9ZKpQSmvsmjRInH48GExadIkUblyZZH0d+QzZcoUMWLECPP
xSnmV5557Thw+fFgsWrSoWHmVX3/9VRiNRvHWW2+JI0eOiLfeesvtyqso7r5bmz8KRU2
eLNsxaZLrrvndd67p0WzfXl6nSMexU9y8aQneL1xwzjV++kmev0T/qN275fWqV3d
O8Prhh/L899yj/rlLsmePvGalSuqW8vj2W3/Xq06maI9fUO295zZxpuX5F/xhduyZEu3b
yPDVryn9e3EFSkhDdu1sCob59hUhIkJMXVq0SYtAg62CpZUsh/u//nFt+SE2u7AEj9+T
UQC8nJ0d88803ol+/fsLHx0e0adNGzJs3T2QU+g24fPlyUbVqXtPXS4ff/yxqFu3rvD
19RWtW7cWW7duNX/tsceeE926dbM6fsuWLSImJkb4+vqKevXqKevXqiXk2prV9/fXXokmTJsL
Hx0dERUWJb7/91q42uZSrQ++wz+YMdFeX8GaGl6d9ftmPuXNddc+9eec2QEOdexxU19Aq
rU0deb8cO55x/9mx5/kGDnHP+om7dsvQUnTyp/9O1Zwtp/Q/qMmeqfuyQFBZa6aZ9/rt45W7e
W55w2TZ1z2is3V/7jBAgRF2f/75SMDCE6dZLPr1FD9hK6k/x8Id57z3rGbNFb795yCIC
Wv08rytN6wMg+Tg30atSoIapVqyaefvppsW/fPpvHXLLyRdSrV8/eU+uWqhaK99HQh/P3
lL6i9e516qVIpf/Q2bXLdNS9ftvxydlbKLivLco0rV5xzjaKUXlq1Aoiixxo517pg5W9q
2lddcuVLd8+bmWsYb7tmj7rnLMn26JTBQw7p1lrFgly6pc86KOHrU8jtl/vzyPmjV9Gp
uXeXzqlZ13T9GFbFpU/EAz2Bw7e8vIrXZE3fYXQJ09uzZZOH/+PD7+PD7++GPceedNo+pVq0
aTp8+XIVSioCDLTMYvv9SmbQ68FwwqMuLbZTbnjmjkwfHWlVQBZFhIukKmfBZFhZU/fMGedcQ5nRGxXk
sEiuuKq1SWLt2crtnj7rnTUiQ68FWqyZLubjSsGFyu3l4cl4cwXl
JE7lEGiBnlpbn1/bNm7Io9rZt8mdlwwbXfz/sYavQtRDqFiImcmd2B3ojRoyAv7NrAFC
JHn1Ubpcvl1PTXS0pSV43IKB8FKG5zBlTX0FMqU+5Mn1T+3EK4rrVJY27Z
yq3agp5RVuftu1/+RbtRIBrD5+cBXXzl2rq1bgR07ZA01Z68qUx4TJwdusiSSY8/Xvr
ay7duAQMHytIpVarIsh9KYO+ulFqkhWle7oLIhZy4qA85Q69eskhsaqr8ZetqSqr8ZetqSqFkW78
8nU2pbeeKQM9VnFlLLyVFFpn28pI9N66i/OHfu1cGRmpxZf08W5RevaLFUO31xhty+8Q
TzismbA8vL1mLrnJlGYR+9JHt43Jy5HJzGbIf/TWrQNiY13b1opQapEq/xwo9dO0rEV
K5EoM9HTGx0euMwtok7519YoYhSkBmLOKJmsR6Dkzdav05jVsKHuPXCU6WgYCmZmWz4u
jsrJkLxigXaA3ZIgMinbutAxfsNdvv8mA1WgEXnhB3fY5okED4N135f4f0pU4p/33Jz5ev
/4QdZ1HntWtkLqBejR8uf7/h4uVV73VEid8ZAT4eU9O1338k/gK6kZaDnytStqyiB3uX
LsvdNTcr4PFembQG5PFTr1vK+Wunb7dtlsFG3ruU9c7WwMJk2BuTQiYpQevNGjHDt56w
8xo2TQfTNm3IpNqU3Ni9PrhDyv//Jfxi+/x7o0UPLlZMeLhcNo09eXS7SFiV9u3RpyWt0lmT
arCz5y9eV3KFHz5MCvSpVLEtOTqQT1OT+nRc+VEDIXa4/QKp21dNX7SFiV9u3RpyWt0lmT
/fmDNGtn+KVPUb5ujDAZg0SI5wWLnTuCVV+T7PniwXL0x0f+cxkXp3VLicgeDPR0yGC
w9Oq5On177JjcMnWrHmelb7WYiKFQe+at1uPzFIMGyV6tI0fkGp32UGa3Dh7s2jGT9qh
Tx7Ju6syZwL33AqtWyZT1118Dfftq2z4isM8DPZ0aPlxuN2wALlxuN2wALlxwzTUzM4Fz5+R9LVO
3588D2dnqnvvGDUvZDFcHes6YkCGENqVVFEqgl5goU66OuHjRssi+kjrVisk9E9Osn79s
zKePYMRkoAcBLL6nfLjXde6/t/W3auLYdRKQOBno61bLGjLIWwYoVrrqkANW
ru+aahQUHy0H+AJCcrO65taxihp3BGiZULF4CrV10/41SU
gVi3H2+Yo5Z+s5ctLL0VS2FtvyeD7vvuAli2d1zY12PaqjmKvTt1q
OzwOcW0tPixp6Cmf06Cm9eQ0ayFmSrmYwqDdOz13Stoq+feU/BMnJwC+/lH18UhLwxRf
y/rRpTm2aKlh3jsizMNDTscTscGD5QzHwHnQnXFnQcX+SsCRlajc8DnDNGT8vxeQo
1xukJIVejANwn0PP3Bx58UN4vT/r2nXfkDNaePeVEKnfHunNEnoWBno7wVrAn07i3vu6J

XT+sePcB5EzK0DPSUnpKUFPXK5WhVWqUwJdBLSKj4OU6elEve+fi4V902JX379deykHB
Jzp8HPvtM3n/5Zee3Sy2sO0fkORjo6ZySvl26tPzjhSrKHQI9V6RuXa1aNcuYx4oW4i1
Ky9IqCiV1e+CArM1WEUratlMnuXKDu+jeXZbFuXIF+Omnko977z05TrFzZ6BrV5c1TxW
sO0fkGRjo6dz99wOBgTJQUVYOcAYhtC2tovDE1C2gbvq28IxbLXv0IiLk5In8fDn7tiL
cbXyewmi0rFBTUvo2LQ345BN5/+WXta3/R0S3LwZ6Olepklx/EnBu+rbwyg1aDsr2xNQ
toO6EjIsXZU+TwQBERTl+vooyGBxL3+bny9Qh4H6BHmBJ337/PXD9evGvz5kjy/a0aSP
XqCYi0gIDPQ+gpG+/+kr9+nIKpTevTh0ZXGpFSd2eO1f62Ch73LypXQ09hZolVpS0bf3
62n6vAMcmZCQmyoA1KMiSBnYnbdrIGao3b8pgr7D0dOCjj+T9l15ibx4RaYeBngfo1g2
44w5ZN+3HH51zDXcYnwcAISFy1mNBAXD2rDrn1LKGnkLNHj13SNsqHCmxoqRtu3eXs8v
djcFgWRKtaPr2449lsNe0KTBwoMubRkRkxkDPAxiNlj84Sr0utblLoFe4lp5a6Vsta+g
p1Byj5w4TMRRKj96xY0BGhn3PddfxeYUpP3cbNlh6hbOygFmz5P2XXipek46IyJX4K8h
DKOnbtWtlz57a3CXQA9Sfeav1+DzA0qOXnOx4+t2devRq1ZLpfiGA338v//Nu3gS2b5f
33TnQa9xY9lrm51uWOFuwQI5pbdAAGDJE2/YREekm0Lt69SpGjBgBk8kEk8mEESNG4Jo
yO6AEQghMnz4dtWvXRqVKldC9e3ccUv4K/q179+4wGAxWt6HKdDodadkSaNFCjlv75hv
1z+9OgZ7aM2/dIdCrVQuoUkWmpB19Xe7UowdUbJzejh0y4K1dW9sJJeVROH176xbw7rv
y8ZQp7plyJqLbi24CvWHDhiExMRExMRERMRHr16/HnHfeeQezZs3CRx99hD1
79iA0NBT33NsnvrheZIjdKpfikpKSYb/Pnz3fmS3Ea5e1Qe/ZtQQFw/Li8r+WqGAq1Z96
6Q6BnMKiTvr10SZb10HrGbWEVGadXOG3r7hMZhgyRbdyxA3j1VVn4OjwcGDlS65YRYRek
k0Dty5AjWr1+PTz/9FLGxsYiNjcXChQuxdu1aHFOmgxYxYhhMCCcOXMwbdo0DBo0CM2bN8f
//d//4caNG1hWZOR0QEAAkNDzTeSyl6W3/QTIIvTxsaRNteSJqVt
AnZm3Sod1ZCQQEOB4m9RQkR49PYzPU9SuDdx9t7z/zjty+fLL7plyhogcpJtALzY2FiEhIYhQVxVRFpR+fC5YFyoaDMc4/NPPiQAkBEHL0+IA69HwBAQmVSDLNnRyMMT7C10WVFnRf+fZCFwdDHqQH
mncbRLrNwOhH2STq3HpgpThtCnFQ1+7znjz8qyIyLkp7GVeRENbO2HwE2wXYr2gs33m5AcfgX
nsfsaqlOQjFtnpdkozRm8Xj2sawGgKkyzf/bVz6gTbmhvPvhaUUjHc6SKTmF+M4S0s1JvL
Zpzl/jH1WkpNpYoLwM7cZwHsmOwrBvgcDD4qKCqDzsx/ZNrjIUDTjSsLIzZjQn6p5gYAr
qJJkgSnm5gkyT0X1ntDnCQ7S3e3tBPc2Ijm7yhunN8Cw+kwNSDWaUxbXWr4+AmhW3WzPSS
6y06fPE3Cx9jX6Nz5t+ZMYSc4pnixf9UGPxtLnLkMAOoHRlmmGdugP+QhrEP13AoxvGnCG
jq4nP8qkHwOrbFQT0krNqN9vnX8Mb/K+ohYqHkpmdDn6bTsMBwDT8sBdJ8WiLq3DYFNZa
5tmTT5eN9TeVxp3BDtFzcCVYvMGfhCcX4tjsfAYYHXr/bJh+NJ8Yvs+5+CgoXoGSvWHUP
dRFsYkc9HnOKRDtbFr9WzB9qhrZVnAmv+c8gRLmeLjOvBjUx4pP+A2gFIkb7NQ/PQ9NpDq
qKp7dbYIi/lE0MRnJtTP9wmcJw0nbBa/c/M2ub1qBhDHMunvfwaRyk1MmZmqqekKGNLgt
TMo/qBdKxfFUVwuTPMItgS4k1gKnI1XoyqLYWX3rsyNRIdkoYW3yF61UC8ZPQf4TQ9NXRH
y7AzlOINQyRG7TkIoIShbxupaIvcYQsExSeRMZ7cVBr/WyPe3hUJjd5/LuiL6vQFU7Y
SXTZ71H+cFH/Gmlr/9SKkDRf0HaRbEaOJkErLXZ6TvFm9nvzepWbEpkMQodU2IsDUEO0
YnoNsRXCBYJ1HMUNKdGwtcrkXPsbdWmePY6QwI4hIRf5Qi/yYKzrZ8sx5IPTcYjz+5xVc
fYzDDS3YP6SBqfIvBuqgChXBEqaq0D4+HFqfy1nI6QDDMXxbLadS//eJlAZTudwlq2HYB
/7t/B4vkAfDKRHSMYbD5JBKOFVQlBLSAoFbWkcjRO07DL0K/mwBm3DcK2bNw/R+bIfUGyV1
1P01p2fiS1K8Dz6r6vHe9SaLJmuq6sJNSBGbEJm2Fa7CsNyq2QE3yMzWaEGOMV4IAmMG
iTuCXHEwwBMkGqQqlL5ofBf5P/dRWkVTd3NHf/ZrN41d/WRbmeFBs6hXGvh1LUe/c
XAtEZD4HkD+MpH8UZmnO7PfkHV+8SKyfJKaXZ7c9lymBbqQrKdTRmtIoTEhBxq51zSDjA
FtHs66fcKcVqBbH8B0fcQTr4qdfyk/Ov/hAhhb9rMjAQDaBcYFnYSVM3ZmBH+W4Ap9VF
ScuTeR+3VbUqfZZlgVZ/v2XeiZJLFDGf3I+BVSzAIONHdZVdXPxaxkmHHd2T0OI+ehbA/j
/vkJwzN5SkMuSVo0ZjtNTEqlEpRbfhtS/kaOiqUvFSbv5T0Rdz3z02u1IsVXf1FkX3NQ
iQt6CqfZtwExEEAozXQhvQoYwULzwa7ToSc2D+4X1W3nMNYXnt3CwqvlYO1ZGxKFN3yfvD
YCyXnThV+9T/dR7RR6yw+7FZExwHaYq7N3KBKpUrXdSYL1C7WzDL3/gNPtc8UvHmx+o
4B7Hh+HYgKUROqjuw5pKSXjNLmE7qtpa1ZeQQ8j5Rn+EStdQqKgfaMZ8iFX3K0Twsg+Y
8vGZ+f9h3I8dfiaqCNiWwvOcxR7i0Gv4jJ8SqwYQfRkhZlKsSCu5sP3lwUR3uYwDChJr
MbdGiBeLi4hAXF4eWLVviiyLrfR07dgzp6enmxy+88AImTm3bsWW5c+ewYcM
GBAYGAgB8fX2xm9OrVC02aNMGECRECRMQFxeHTZs2weg0OsryMcHUDLijtbUO3lSDow
PDJTrzLoLtWbeulOgBzhWYkVJ27rb+DxAlnpR2lVa+lZvadvCwsPlurzsySMid6KbIgD
Vq1fHl2VELaLINFODwYDp06dj+vTpNo+PiiiA1q1b1WqiW3n0Ubmw+nffAfPXyPAmPVbyHR0n

bNmmi/Ri2wjwxdQs4NvPWHUurFNauHXDggEzfDhxY/OsFBZYePT0GekRE7kg3PXpknw4
d5CzOrCzgf/+r+HnccXwe4JmpW8CxmbfuOhFDUVbh5AMH5BjDypXl55eIiBzHQM9DGQy
WSRmOpG/dPdBLTpalLCrCnWroKRxJ3bpraRWFUmIlIcF2jUclbdutG1eVICJSCwM9DzZ
8uNxu2GAJaOzlroFe7dqy/EhurhzgXxHuVENPUTjQK1qbrTSXL1u+x+4241bRsqUM4K5
cAU6dKv51PY/PIyJyVwz0PFijRjIFVlAArFhRsXO4W7FkhdFoWXaqoulbd6qhp6hbVwa
wN2/aF8AqvXl16wJVqjinbY7y9QVatZL3i5ZZyc6WazQDDPSIiNTEQM/DOZK+vXYNuHh
R3m/USLUmqcbRmbfuNj4PkEGeMtHEnnF67j4RQ1HSOL1du4AbN4BatYDmzV3fLiIiT8V
Az8MNGSKDh4QE4OhR+557/Ljchoa6Z00zR2feumOgB1RsnJ47l1YprKSl0Aqnbd2ld5W
IyBMw0PNwNWvKAsoAsHSpfc911/F5Ckdn3rp7oOfJPXp791pPouH4PCIi52CgdxsonL6
1Z4C/uwd6So+eJ6VugYqVWHH30iqK6GhZPiUryzL+Mz0dUFY3vOce7dpGROSJGOjdBu6
7T65skZQE7NhR/ue5e6Dn6T165U3dXrkil7sD3D/QMxqB1q3lfSV9u2WL/AekcWOgTh3
NmkZE5JEY6N0GAgKABx+U9+2ZlFF4VQx3pARoZ87Y11MJuGcNPUXh1K2tenNFKWnbiAg
Z0Lu7ouP0mLYlInIeBnq3CSV9+9VXspRFWYRw/x69O+4AvLzk67G3TqA71tBTREbKCQk
ZGUBaWtnHu3uh5KKKzrxloEdE5DwM9G4T3bvLIsNXrwI//lj28ampQGamDKTq13d68yr
Ex8eygLy96Vt3rKGn8Pe3vK7ypG/1MhFDoQR6+/cDp0/L2eBeXvIzSkRE6mKgd5swGi0
rZZQnfasMlI+MdO/lqCpaS89dx+cp7Jl5q5eJGIoGDWQvanY2MGeO3Ne2LVCtmpatIiL
yTAz0biNK+nbNGlkMuTTunrZVVLSWnicFenpL3RoMlnF6CxfKLdO2RETOwUDvNtKyJdC
iBZCTA3z9denH6iXQq+jMW3cP9MpbYuXaNeD8eXnfXde4tUVJ3968KbcM9IiInIOB3m2
mvEui6S3Q89TUbVlj9JTevPBwwGRybpvUpAR6gByTGBurXVuIiDwZA73bzCOPyNTZtm2
lB0d6CfRu99St3sbnKQoHsLdu2b9qCxERlQ8DvdtMRIRlduOyZbaPycuz/CF290CvcI9
eeWrOAe5dQ0+hzHROS5MrR5REb+PzAODsWeDFF633PfWU3E9EROpioHcbUtK3X3xhOzh
KSpLBXqVKljIf7ioiQvZQ3rwJXLpUvue4cw09RWAgEBIi75eWvtVbaRUAOH68eIHr/Hz
7lnwjIqLyYaB3G3rwQTku6sgRIDGx+NeVtG2jRrK+mTvz9ZX1AYHyp2/duYZ2en+o
xdWvrc2U0Wl4vERGpx83/jFtcvXoVI0aMgMlkgslkwogRI3CtjBoh3333HXr27ImIiAgo
YDAYk2ohqsrOz8eyzzyI4OBiVK1fG/fff3bU3U3K0Mv4PIW9M2/dfXy
eoqxALz0dOHdO3tdToBceBceDixYIIM7QG7nz3f/3mMiIj3STaA3bNgwJCYmYmUWIFfvnlF2RmZQJ///7Iz89X/yhy
SExMxYsSIUp+TlUp746233YWFzp0746233irrTNNp/Ucf23fT3T6/F2RmZQJ///7Iz89X/yhy
4FSV9u2yZZ+Mm7cWPhrL1kugp0zIKO/MW70EemWVFHStrVru28fm9W7P2zcgKfbjhCIm6crXW
LiIg8k7fWDSiPI6ePI0eOHMjNzUVkZCSyc3OhaIUcq0haWmerpBqudKi+NlwqDGBrPEi4Dfmdk8ENh9fT5mDkmXfE9PV2x5vFaPS+K/GOwvGCJk1X
0dCxatAhffPEEv5dyOvLL79EOjEm/ERE+QENmREk3+tnxF69nZ/++hLYQejQHe7l5Ax4Mvfo
eeyirxIoeJ2IUFh7OXjwiImfTRY/ezp07YTKZHMkBRo2nkvMr2LBdzjZ9HgCqz8X7H3+2JM7KeTT2gSwX1DjoSB6k+9HP6vR
cxBWKcmrXro3mzXuXet7s7GxkZDRY3fTG1xcYMkTeiGTmpC45VKwd9HENVE8mqH9IrDRDJvt9xWfwdBdUWEx6hzMLQXh4ddtPvQJzTaIWk
LF4FeamoqatWqVx/rVq1kJqYdOTHnowwvVQohYkJbzTauatJzWsaL7fmTuyAtWWO6vrP/g0t5gYGhlHKNEK0iM
WlPTdvkUFPQpBUb8eoSCDRiU0i/f8PhwmE7bkt3+0rsnyjMXZmWuxuIq9c7wMKUaB5xFr8wTNWuCIf3DM3NE1hM0L3b2zzgVRxUTPdl0lwWMO8G0M4XXGRvodbCuB1i85cOsX5NmD4+qX+LQCA1s35M3X00VR5VqyWGCpn+m9kWyPx9
tfUO5NwpxLlyBGFFeyLoVlBxaU0iJRdh6OHt9Exek2yD8+SUUJvE5ZxgGa6zp+1KkMO6ilNNQvzX9HlUd5twv8XIrHs/ZNERLo1J5KNnWxKVpfv0K82kmAetpy+0M
YKBHREQl0bRHLzg4GFFRUa/P39ERsbi/T0d83N9++w3p6emlBmRladOmtx/2H4RqRVz8Dp6flX0NrNGvY6Aj3pyVjafZMG3G9ofL6z2f4TDA
8fLBx40bzvpSUFBw8eNCh8+rJ8GD7OnSY6Dn7w+Ehsr7ZaVv9VJDT1HSOL0jR+Q
2LMzS60dERFSULsboRUdHo3fv3SdEZz3fv3hg7dix27dqXbt2YezYsej0J/VjNuoqCisWrXK/Pj
KlStITEzE4b9zXMeOHUNiYqPJ5/J3J3JZMLo0aPxz3/3b8a+ffvw6KOk2UNp6ZwvlTrR8a+ffvw6KOPokWLFuZZZJuJ6
uUSOgQwe5MsGKFZZAr4SJzG6rvDNv9TI+T6GM0ys6ZpYTMYiIqDx0EegBwNWNKlS9RI9GiRQ

ExcUhLi4OLVu2xBdffGF1zLFjx5BeaGHQ1atXIyYmBv369QMADB06FDExMfjkk0/Mx8y
ePRsDBw7E4MGD0blzZwQEBGDNmjUwKtVcbwPKpIwvv9Rnjx5Q/pm3egv0SurR40QMIiI
qD13U0QOA6tWr40tbyzgUIoqMxB81ahRGjRpV6nP8/f3x4Ycf4sMPP3S0ibo1ZAgwaRK
QkCDLrgD6C/TKO/PWUwI99ugREVF56KZHj5ynZk2gd295PydHbvW27qinp27PnLF8bwD
OuCUiovJhoEcALOlbQAZ+V65o15aK8NTUbUgIULkyUFBgafv16zLwAxjoERFR6RjoEQD
g/vsBpWLMpUsyFbpokbZtskfh1JtfT0VENPYTAUT98qM25DQvRT65CIiLTBQI8AyB6
8wqnBggLgqacsq2S4OyXQy8gArl2zfYzeaugpigZ6nIhBRETlxUCPAADHjxfvCcvPL3m
dVXcTECBTzkDJ6Vu91dBTFC2xwokYRERUXgz0CICsp+dV5NNgNOprUkzL0Nj5PwR4
9IiKqKAZ6BAAIDwcWLJDBHSC38+fL/XpR1oQMTwn02KNHREETlpZs6euR8o0cDvXrJgKJ
hQ30FeUDZtfT0GugpqdvTp4H0dEsgy0CPiIjKwkCPrISH6y/AU3hq4zCF
Za9UCgoO1bRcREbk/pm7JY3hq6tbLC6hfX97//nu5ZW8eERGVBwM98hilpW71WEOvMCV
9+8MPcsuJGEREVB4M9MhjKIHe1auynl5heq2z4Krfs0SMiovJgoEceIzDQslJ
E0fStXmvoKYqWuWGPHhERlQcDPfIoJaVv9VhDrzDPfIoJaVv9To+T1E00GOPHhERlQcDPfIoJc281Xugp4z
RA+RsW2UVECIiotIw0COPUtLMWyXQU3r89KZuXcvKJXparYSIiLTFQI88iqembj
okPd/+w1YtEjb9hARkT4w0COP4omp27NngSeftDwWAnjqKbmfiIiIoNAz0yKPYSt3qvYb
e8eOW3jxFfr5l7VsiIqKSMNAjj6KbtPSgKwseVV8GmXIxo1sozPUxiNHKt
HRERl002gd/XqVYwYMQImkwkmkwjRozAtWvXSn3Sn3Od999h169eiE4OBgGwGJiYnFjun
evTsMBoPVPP1+/axITEZHr6CbQ6wzuALmdP1+/axITEZHr6CbQGzZ
sGBITE7F+/XqsX78eiYmJGDFiRKnPycnJQefOnTFnzpxSj4uNjUVKSop5++CDD9RsOhERkRrMwbPjuFP1/NppO
LFZ2QoefxeYrRo+XriI+X29GjtW4axc6dOgMkTLTYP369idi1xwK4i1aDjnm3Le2fCF
iY2Nx7NgxNnSxObzlEAwydbip4UEBAQgDNDRU1TaTdurVA/bv9ezZ99e0OzFIyIe+i
iR2/nzp0wmUzIA8AOnbsCJPJhB07/qVLlyI4OBjNmjXD888/j/888fHZ2NjI
yMqxu5D6KTsjwlECPiIjIXrro0UtNTWtWrWK7a9Vqg++sM1WrdzOt5zSmN0R6Qw
4EFOnTsX+/fuxcePGPjgcRmGbSj7x+g6j7dz5h3
AGGyMohdC2Nxvj7Fjx6Jnz55o3rw5hg4dim+++QabNm3C/fuxbS0f2r84FSu3ej6sYrkcHQB+MM8XsFNz93Gd25QPbyN4
wjV4g5qaO1nybCDbxNn3L11WpAOy04QLpEDlU0TgwFFuWpR77VSgC4I5xsWDt4e1Gs
3sAQmH4U3WOKzUvm6GRWoOPTOOP6EHmLUSlaxNeW8Od9o7XDhHSAlUk6Htr
t5n8uv0MFOHm8OpjF0iHn1ycNXWdLH0kNLz+2XVE7N0h99JkEqUs1gT5FAyOHjHLHU
DNsuZH4e+MVJQMWnP0GwJSMrrhKeOYQYmuBFlBFY8ehk5bP4Q+eZkSrnNmAGpY2i9nOA
tbKX+CYcNIpq+IqOH+9CEpRIvy4aeocdOqRtAqNKDl5vC6nvhaJKp5J0PmPjlHmCf7Vi
GN4npCuz6gP1j6XE/8ovDZNvhSzzT9wbBkzoBQkRqTvkBoSzwaEDh9fH77IQ8COd1FvpTUo5cp3t4SwIyHN6q8cCHnAQMTrb6fkIq2mymjqkzKI3mN40y3
8qQWGAVjc+zPnlVYQ4g8UzNv7WN5cmaFoLNu3GWjGIvZMwHYNzBR7eHGYJr2
i9Cgb5zrHvdRsLnRozM8T+gXSwHh7DD6gHq3rFKaYcpE+rHgLCCfD3vxVvnHzrYk
gDCvFw3EFQO3xaK0PzDGzNvKDyDRzfP6G4vN2WqS+4Hpn54Mwv2TQbH4jEo5pkKoVnmqMe
ExcUhLi4OLVu2xBdffGF1zLFjx5BeaGHQ1atXIyYmBv369QMADB06FDExMfjkk0/Mx8y

9qzi+dxXD963i+N5VDN+3irsd3jtOxiAiIiLyUOzRIyIiIvJQDPSIiIiIPBQDPSIiIiI
PxUCPiIiIyEMx0NOBuXPnIjIyEv7+/mjTpg22b9+udZPc3rZt23Dfffehdu3aMBgM+N/
//qd1k3Rh5syZaNeuHQIDA1GrVi0MHDgQx44d07pZujBv3jy0bNnSXHg1NjYWP/74o9b
N0p2ZM2fCYDBg0qRJWjfF7U2fPh0Gg8HqFhoaqnWzdOQ+SFhoaqnWzdOHcuXN49NFHUaNGDQQEBODOO+/
E3r17tW6WUzDQc3MrV67EpEmTMG3aNOzbtw9dunRBnz59dCcZqEeHuTLG4ZGVlad00txceHo633noLCQkJSEhIwN1
3340BAwbg0KFDWjdNN/bs2YMFCxagZcuWjdFN5o1a4aUlBTz7cCBA1o3ye1dvXoVnTt
3ho+PD3788UccPnwY7733HqpWrap1O5yC5VXcXIcOHdC6dWvMmzfPvC86OhoDBw7EzJk
zNWyZfhgMBqxatQoDBw7Uuim6c+nSJd2SqVQtbt25F165dtW6O7lSvXh3vvvvsuRo8erXV
T3F5mZiZat26NuXPn4v4vXXX8edd96JOXPmaN0stzZ9+nT873//Q2JiotZN0ZUpU6bg119
/vW2yY+zRc2M5OTnYu3cv4uLirPbHxcVhx44dGrWKbifp6ekA8l1RUpl3OxMcfnCQ44MbbZAs
LsbGxWjdHF5555hn069cPXv21LopunL8+HGUrl0bZEuvTtCWtWmJisHDhQq2b5TQM9NxwYWloa8vPzERoaarU/NDQUqaiq8Pm4KefxERGVKKbmanwY9Nmzwefd6eXuJTtEg7sk6tbVN81fcNl4TV9jxwOwR7nAU7WGX4dMlVwh6bJA7XbBZmoQvu1D
Q2JiotZN0ZUpU6bg119
ulRpXFzYA3G1uv3SUi0Ak1A9NPT6e4SGKVx1zHlGoNa/zhF2nf8ZOzBqNNQd4fbVB4pzQU71LHtaf0T9q/AB3gcXGA4LTz/ecVLlGYeVhCxsWdrd9A3T3QZBpMiKfAjZFmK5xoxEDbXLWXIYGzYeahmbvPtVixxjgZzSC+qzNvrNzO/0ZeVO8nJMdGZk6eiZkw/7yw03RNtuK4h3GGf7BUbmD3vV+ZFYPnU8wF3svfSjV0CQTvSCh0W/SlaRh7WvhITtk7i1yYXdaicu3vU3C8zEXJaK5utOWgI2OScY8J7YLmUPsm0HA8JXmCwd/RLwxsIQkW5c1fVpAPcYd7U6Ny5dazdcGpMhQBDJVdyV7iLW3bT4MD9lp8ABNQ1zl0/T3x9ozZNB2uzxsZlGo8PrQTe6UZXyA4LQ6oHnB4nsQ9/8c+Bxa0YaRWH1dRm2s11D/kzkULdLQQWDHJCuJ7+dYqBnmrdH6l4tVjvCp6xP2KYm0cvgyE8M9ZqDsAKLq8Sxhqq7NjUJZphJRtTeMKgJcn3ay+GXR7vnGS8S5UaGk6Ry0R1o8g5cd4Rbv1B6IlNnzz/zpvGeJBMyPH9VhP1rWbBkrt5bB7GdlgApIBm0yZNZdsC6rTGMPbqG5pCBh+LVmBK/d/4dkbZoOtFCrvFuYxCP/BQZqdp0kRnVewS0fxzTrYgC/l33AhTaVWpJH8OgTe+qE/bdO3D2hTt4sL/A5s0b0qR5e/A6D9OdZXL2H8hRUqRaJIDZPsLQWNsz1QD4e5tGrimNFBEKYFEgi9z95zeURhnkmSkNFdyCEY2o9Ff2AX9EJI2EEr6+DoIOJSlUhqbp/33NnNTVTeOX2nHiHpZ+SuAh4u31Aqd+Yrx/ABgsM90DQlVSVPcN8FSxNoHhbMQaOaGBr84+vREWbsM1eHMsG5lRg/qZ2YWx8xGnkP5HCRLxLpOOxMJ1BzTy5JlTHHzIiREFKbIY6jHUu6eVeEm5QwV93gGGsyNKUEfuRHY8FSN2pIFmvHAATsCOUsJDR8B5OcqMThhHJu2iNEH2jAfIGPzEFGtZTxw1hbuw5C/HPfuT8sjUdwG2cZ7OdYdCWCP8MXEwNgwxHYo2H1Zd9nt8q2dVCRV0RIAqQhKLm4ZGqvyk/Qcx62/WIr7/I6NVv1M1DJj0Q4nKNwm9mw/nQ37dLP9HcgOxpGeqsF2+WC/JPzKcaqwixRA1W5kLGJ7mEp8xNlIsZcT2vyrhdRO4ySXuP8ZK3JnqN2W1s3bDGPdeW3W2JD7iO0jEfrUhJWWV7ObOSjkaHnLDS2m56G/8bFAiXvvngsEW54l9S+mX5vl/O5j74Mue0i29cexZsBFEQBXfJWSvzxQ3CTvv9tqxDMmafrcNt19Is6Rp4JVAtc7x9wtj+Z3ZIXw7hw04gPfYuywGq/8C8NZLeXEpNIoVyqmUWlSWJ2GXqNyLxJPk+sd0tEOvIQM5a69F1prm+FpUyrVs9qjk+TCztyNJQQZ8MeFBBzCGk8NLk4F1/4LmMvP5xJ1p5cFiTrCWtnmMuqH/OuLPEU/7Dgw5zvDbT5FNaI+aN2TfJUaP0UE4V3D0knQWY8EGZsxMv5Rba0Dvqvi9HWxMp/2ZmlZ6xY8qwyx5jB0eHyThKVmcGxU0kNTbtFDPNrA1xxf2cBEyjRm4KVZcWkV+cncQHKuhe4jGJ2pN3rGPiuJmMjyOrVv+GH8n5D9cCHpVrKp+ZGwdyx2+9c1wDxTgOIw1UNNjRnbbw0yH1Xd7zuSSz9UjTuEyuGFFbZwqixyxpONz9G7pqIaV+N7sW/gxZJe3YdlPSlYqYPxSD/GE5UkOwM0/MLVi/VuNfwfBTeiZLaTe5tHlGNb+rpXSpLhKftOZPwgeS0SfbFAFJFpXeMfnNuVBbdbQEXZlGBTj9fC2eFtZYU8GL3EXMeKXJ4qjhzvxPiU7jGeDs4dyIkyURxA8EBpmQkHSdTqmDjrBS+EVq3EHnVvhnJiOdHMptvHM8dXyZuKWN8Qk6rxv6aCxEGSFZWj28i/3QX1Vn3B3jwBJr8cgJGIx9IU4vPKURzPEzkGk3bqh0ZhBYaj3wDWY0TEt8IPy5NpEuRCI6crFHd1qeGFPY2Ug3UPWsAgBrU/m8NVXoDDEQLlvwYY2dsq+f7vxFx7dTYEefg6WYIlJnG27DSL09w3MSzZ5j2DTVdR6eEKKZImAG3/W54m/2QKDXEOIlsyMgDNZz3JGi+PxbE4tCgIgaVa3Gy4Bsr90V5k/CbQBgbZ1M6EkbLnygsPL8Fvrx/SpcyqnkkBg2SVe2lJAycxF6aKgApvzr8AAAAAElFTkSuQmCC

```
     ]
    },
    "metadata": {},
    "output_type": "display_data"
   },
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "Done playing.\n"
    ]
   }
  ],
  "source": [
   "read_and_play(\n",
   "    csv_name=\"ch5_damped_sine.csv\",\n",
   "    base_freq=220.0,\n",
   "    note_duration=0.3,\n",
   "    samplerate=44100\n",
   ")"
  ]
 },
 {
  "cell_type": "markdown",
  "id": "93b400ca",
  "metadata": {
   "pycharm": {
    "name": "#%% md\n"
   }
```

},
"source": [
"## Part C: Interactive Sliders\n",
"We’ll let you pick:\n",
"- `x_max` (domain limit)\n",
"- `num_points` (how many samples)\n",
"- `alpha` (pitch scale)\n",
"- `base_freq` and `note_duration` for playback.\n",
"\n",
"Each time you change a slider, the code regenerates the function, snaps the pitches, writes a CSV, reads it, and then plays the notes. We also plot `(x, y)`."
]
},
{
"cell_type": "code",
"execution_count": 7,
"id": "fc9149b3",
"metadata": {
"pycharm": {
"name": "#%%\n"
}
},
"outputs": [
{
"data": {
"application/vnd.jupyter.widget-view+json": {
"model_id": "7905130ab101451dacbe31cd779fabe4",
"version_major": 2,

```
      "version_minor": 0
     },
     "text/plain": [
      "interactive(children=(FloatSlider(value=6.28, description='x_max', max=10.0, min=1.0, step=0.5), IntSlider(val…"
     ]
    },
    "metadata": {},
    "output_type": "display_data"
   }
  ],
  "source": [
   "@widgets.interact(\n",
   "    x_max=(1.0, 10.0, 0.5),\n",
   "    num_points=(5, 50, 5),\n",
   "    alpha=(5.0, 25.0, 1.0),\n",
   "    base_freq=(110, 440, 10),\n",
   "    note_duration=(0.1, 1.0, 0.1)\n",
   ")\n",
   "def interactive_damped_sine(\n",
   "    x_max=6.28,\n",
   "    num_points=20,\n",
   "    alpha=15.0,\n",
   "    base_freq=220.0,\n",
   "    note_duration=0.3\n",
   "):\n",
   "    # Step 1: Generate & write CSV\n",
   "    _x, _y, _p = generate_and_write_csv(\n",
   "        x_max=x_max,\n",
```

```
  "        num_points=num_points,\n",
  "        alpha=alpha,\n",
  "        scale=[0,2,4,5,7,9,11],\n",
  "        csv_name=\"ch5_damped_sine.csv\"\n",
  "    )\n",
  "\n",
  "    # Step 2: Read & playback\n",
  "    read_and_play(\n",
  "        csv_name=\"ch5_damped_sine.csv\",\n",
  "        base_freq=base_freq,\n",
  "        note_duration=note_duration,\n",
  "        samplerate=44100\n",
  "    )\n",
  "    print(\"Done interactive call.\")"
 ]
},
{
 "cell_type": "code",
 "execution_count": null,
 "id": "d0081498-1d89-4ac8-afa8-09d3a212980a",
 "metadata": {
  "pycharm": {
   "name": "#%%\n"
  }
 },
 "outputs": [],
 "source": []
},
{
```

```
   "cell_type": "code",
   "execution_count": null,
   "id": "e17af64a-31fe-412b-9ed4-1fece640ca7d",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.21"
```

```
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

xValue	yValue	PitchSemitone
0	0	0
0.34210526315789475	-0.195919422	-3
0.6842105263157895	0.26751545016919204	4
1.0263157894736843	-0.266436186	-3
1.368421052631579	0.22884315620042545	4
1.7105263157894737	-0.178056753	-3
2.0526315789473686	0.12767641129352963	2
2.3947368421052633	-0.084506422	-1
2.736842105263158	0.050976857	0
3.0789473684210527	-0.026973135	-1
3.4210526315789473	0.011112861240602272	0
3.763157894736842	-0.00156627	-1
4.105263157894737	-0.003467668	-1
4.447368421052632	0.005525044	0
4.7894736842105265	-0.005794696	-1
5.131578947368421	0.005124967	0
5.473684210526316	-0.004074458	-1
5.815789473684211	0.002977926	0
6.157894736842105	-0.002010651	-1
6.5	0.001243087	0

Chapter 6: Rhythm, Meter, and Polyrhythms (Optional)

Preliminary Concepts

February 9, 2025

6.1 Introduction

In previous chapters, we focused primarily on **pitch** (along a helical axis) and **timbre** (one extra dimension) for sonification. Chapter 6 explores **rhythm**, **meter**, and the intriguing concept of **polyrhythms** as additional ways to encode or highlight data.

6.2 Why Rhythm Matters

Temporal Structure When notes occur in a steady beat or metrical grid, listeners more easily perceive patterns as *musical gestures*. Rhythmic frameworks can also make it simpler to interpret the timing of data events (e.g., the progression of a function over time).

Basic vs. Complex Meters

- **Simple Meters** (4/4, 3/4, etc.) create a regular pulse, dividing time into equal measures and beats.
- **Polyrhythms** occur when multiple pulses or subdivisions run simultaneously at different rates (e.g., a 3:4 ratio).

6.3 Data Encoding Through Rhythm

Mapping Data to Rhythm Instead of mapping data just to pitch or timbre, we can map it to:

- **Note Durations**: a larger data value means a longer note, smaller data means a shorter note.
- **Onset Timing (Events)**: data crossing certain thresholds could trigger an extra beat or skip a beat.

Polyrhythm as Extra Dimension A single data stream might drive *one* rhythmic pulse while a second data stream drives *another*, creating a complex interplay that can represent multi-dimensional data.

6.4 Simplifying Polyrhythms

Example: 3:4 Polyrhythm We might run a "3" pulse and a "4" pulse together, each triggering notes in different instruments or pitch ranges. Over the course of one measure, the pulses align at the start, diverge in timing, then re-align at the measure's end.

6.5 Conclusion

Rhythm and meter are powerful tools for making sonification feel more *musical* and for encoding additional data dimensions. Polyrhythms, in particular, can represent parallel data streams or highlight complex relationships. In the next (expanded) discussion, we'll delve deeper into various rhythmic mapping strategies, usage tips, and a step-by-step example.

Next: The Expanded Discussion for Chapter 6, exploring polyrhythms and practical code approaches.

Chapter 6: Rhythm, Meter, and Polyrhythms (Optional)

Expanded Discussion

February 9, 2025

6.1 Detailed Look at Meter

Simple Meters A meter like 4/4 means four beats per measure, each beat typically receiving one quarter-note's worth of time. The sonification might place each data point on consecutive quarter beats, or compress/expand time for a faster/slower tempo.

Compound & Odd Meters Meters like 6/8 or 5/4 can give a different feel—some data might be mapped to a 5-beat measure to highlight irregular cycles or phases.

6.2 Polyrhythms: 3:4 Example

Definition A polyrhythm is formed when two (or more) rhythmic streams are played simultaneously but have different beat subdivisions. For instance:

- Stream A divides the measure into 3 equal segments,
- Stream B divides the same measure into 4 equal segments.

These two streams only align at the measure's start and end, creating interesting rhythmic interference patterns in between.

Data Encoding Strategy

1. **Voice A (3-subdivision)** might represent *data series 1* (e.g., pitch in a certain range or timbre).
2. **Voice B (4-subdivision)** might represent *data series 2* or a second dimension.

Listeners hear the interplay, potentially perceiving relationships between the data sets if the pulses clash or align at certain points.

6.3 Practical Implementation Tips

Tempo and Looping When using polyrhythms for sonification, it's common to loop over one measure repeatedly or progress measure by measure:

- If the data is large, break it into chunks—each chunk maps to one measure.
- Or let the measure represent a time slice in real data.

Instrument Variation Using different instrument timbres or pitch ranges for each polyrhythmic stream can help the ear separate them. For example, a bell-like tone for the 3-subdivision, and a bass tone for the 4-subdivision.

6.4 Example: 3:4 Polyrhythm with Sine Tones

In the forthcoming code:

- We'll define a measure length (e.g., 2 seconds),
- Subdivide it into 3 equal triggers for one voice, 4 for the other,
- Each voice can map different data arrays to pitch or amplitude.

Advantages

- Clear demonstration of polyrhythm alignment (the two voices only line up at measure boundaries).
- If the data is correlated, the streams might converge more often, producing interesting patterns.

6.5 Limitations

Listener Overload Too many polyrhythms (e.g., 3:4:5) can be overwhelming unless carefully spaced or assigned to distinct timbral "channels."

Measure vs. Real Time Mapping data into polyrhythms implies a "musical time" approach rather than real-time streaming. For real-time data, polyrhythm might be best if the data itself inherently cycles or we artificially slice the data into measure-sized chunks.

6.6 Conclusion

Polyrhythms offer a powerful extension to the Helical Sonification System, allowing multiple data streams to unfold in parallel while maintaining a cohesive rhythmic structure. In the accompanying Jupyter Notebook, we demonstrate a simple *3:4 polyrhythm* example with optional pitch or amplitude mapping for each stream. Experiment with tempo, subdivision, and data arrays to discover new ways of hearing your data sets.

Chapter6_Rhythm_Notebook.ipynb

```
#%% md
# Chapter 6: Rhythm and Polyrhythms (Revised)

In this notebook, we demonstrate a simple **3:4 polyrhythm** approach without using
`sounddevice.get_stream_time()`, since it's not supported in some versions of `sounddevice`.
Instead, we accumulate times ourselves.
#%%
import numpy as np
import sounddevice as sd
import ipywidgets as widgets
import matplotlib.pyplot as plt
print("Chapter 6 polyrhythm notebook (revised) loaded.")
#%% md
## Updated `play_polyrhythm()`
We:
- Compute times for stream A (subdiv_a triggers) and stream B (subdiv_b triggers).
- Merge them into a single sorted list of events `(t_event, freq)`.
- We keep an internal `last_event_time` and do `sleep(wait_time)` to schedule.
- We never call `sd.get_stream_time()` or `sd.wait()` for time alignment beyond note playback.
#%%
def play_polyrhythm(
    measure_duration=2.0,
    subdiv_a=3,
    subdiv_b=4,
    freq_a=440.0,
    freq_b=220.0,
    data_a=None,
    data_b=None,
    alpha_a=0.0,
    alpha_b=0.0,
    samplerate=44100
):
    """
    Plays one measure of polyrhythm:
      - Stream A with subdiv_a triggers
      - Stream B with subdiv_b triggers
    measure_duration: total length of the measure in seconds.
    freq_a / freq_b: base frequency for each stream.
    data_a / data_b: optional arrays for offsetting pitch. Must have length >= subdiv_a or
subdiv_b.
    alpha_a / alpha_b: how strongly we map data to semitone offset.
    """

    # times at which Stream A triggers
    a_times = np.linspace(0, measure_duration*(subdiv_a-1)/subdiv_a, subdiv_a)
    # times at which Stream B triggers
    b_times = np.linspace(0, measure_duration*(subdiv_b-1)/subdiv_b, subdiv_b)

    # If no data arrays are given, default to zeros.
    if data_a is None:
        data_a = np.zeros(subdiv_a)
    if data_b is None:
        data_b = np.zeros(subdiv_b)
    data_a = data_a[:subdiv_a]
    data_b = data_b[:subdiv_b]

    # Build an event list: (time, 'A'/'B', freq)
    events = []
    for i, t_a in enumerate(a_times):
        pitch_offset = alpha_a * data_a[i]
        final_freq = freq_a * (2.0 ** (pitch_offset/12.0))
        events.append((t_a, 'A', final_freq))

    for j, t_b in enumerate(b_times):
        pitch_offset = alpha_b * data_b[j]
        final_freq = freq_b * (2.0 ** (pitch_offset/12.0))
        events.append((t_b, 'B', final_freq))

    events.sort(key=lambda e: e[0])

    # beep_dur is how long each beep lasts
    beep_dur = measure_duration / (max(subdiv_a, subdiv_b)*2)

    last_event_time = 0.0
    for (t_event, stream_id, freq) in events:
        wait_time = t_event - last_event_time
```

```
        if wait_time > 0:
            sd.sleep(int(wait_time * 1000))
        last_event_time = t_event

        # generate a short beep
        t = np.linspace(0, beep_dur, int(samplerate*beep_dur), endpoint=False)
        wave = 0.3 * np.sin(2.0*np.pi*freq*t)

        sd.play(wave, samplerate=samplerate)
        sd.wait()

    # end of measure: if there's leftover time
    leftover = measure_duration - last_event_time
    if leftover > 0:
        sd.sleep(int(leftover*1000))
    print("Finished one measure of polyrhythm.")

print("play_polyrhythm function is now revised.")
#%% md
## Single Measure Test
Here, we do a single measure with 3:4 subdivisions at measure=2.0s.  Frequencies: A=440, B=220.  No
pitch offset.
#%%
play_polyrhythm(
    measure_duration=2.0,
    subdiv_a=3,
    subdiv_b=4,
    freq_a=440.0,
    freq_b=220.0,
    data_a=None,
    data_b=None,
    alpha_a=0.0,
    alpha_b=0.0
)
#%% md
## Interactive Sliders
Let's let you vary measure duration, base frequencies, and the alpha offsets (pitch offsets from
data). We'll use small arrays for `data_a` and `data_b` so each sub-trigger can differ in pitch if
`alpha_` is nonzero.
#%%
@widgets.interact(
    measure_duration=(1.0, 5.0, 0.5),
    freq_a=(110, 660, 10),
    freq_b=(110, 660, 10),
    alpha_a=(0.0, 12.0, 1.0),
    alpha_b=(0.0, 12.0, 1.0)
)
def interactive_polyrhythm(
    measure_duration=2.0,
    freq_a=440.0,
    freq_b=220.0,
    alpha_a=0.0,
    alpha_b=0.0
):
    data_a = np.array([0, 2, 5])  # for the 3-subdiv
    data_b = np.array([0, 3, 6, 9])  # for the 4-subdiv

    print("3:4 polyrhythm => measure=", measure_duration,
          "freq_a=", freq_a, "freq_b=", freq_b,
          "alpha_a=", alpha_a, "alpha_b=", alpha_b)

    play_polyrhythm(
        measure_duration=measure_duration,
        subdiv_a=3,
        subdiv_b=4,
        freq_a=freq_a,
        freq_b=freq_b,
        data_a=data_a,
        data_b=data_b,
        alpha_a=alpha_a,
        alpha_b=alpha_b
    )
    print("Done interactive polyrhythm.")
#%%
```

{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "b0af16d0",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 6: Rhythm and Polyrhythms (Revised)\n",
    "\n",
    "In this notebook, we demonstrate a simple **3:4 polyrhythm** approach without using `sounddevice.get_stream_time()`, since it's not supported in some versions of `sounddevice`. Instead, we accumulate times ourselves."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "7d0e1ee5",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {

```
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Chapter 6 polyrhythm notebook (revised) loaded.\n"
     ]
    }
   ],
   "source": [
    "import numpy as np\n",
    "import sounddevice as sd\n",
    "import ipywidgets as widgets\n",
    "import matplotlib.pyplot as plt\n",
    "print(\"Chapter 6 polyrhythm notebook (revised) loaded.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "b0c59d00",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Updated `play_polyrhythm()`\n",
    "We:\n",
    "- Compute times for stream A (subdiv_a triggers) and stream B (subdiv_b triggers).\n",
```

```
    "- Merge them into a single sorted list of events `(t_event, freq)`.\n",
    "- We keep an internal `last_event_time` and do `sleep(wait_time)` to schedule.\n",
    "- We never call `sd.get_stream_time()` or `sd.wait()` for time alignment beyond note playback."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "8ac56ede",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "play_polyrhythm function is now revised.\n"
     ]
    }
   ],
   "source": [
    "def play_polyrhythm(\n",
    "    measure_duration=2.0, \n",
    "    subdiv_a=3, \n",
```

```
    "    subdiv_b=4,\n",
    "    freq_a=440.0,\n",
    "    freq_b=220.0,\n",
    "    data_a=None,\n",
    "    data_b=None,\n",
    "    alpha_a=0.0,\n",
    "    alpha_b=0.0,\n",
    "    samplerate=44100\n",
    "):\n",
    "    \"\"\"\n",
    "    Plays one measure of polyrhythm:\n",
    "      - Stream A with subdiv_a triggers\n",
    "      - Stream B with subdiv_b triggers\n",
    "    measure_duration: total length of the measure in 
seconds.\n",
    "    freq_a / freq_b: base frequency for each stream.\n",
    "    data_a / data_b: optional arrays for offsetting pitch. Must 
have length >= subdiv_a or subdiv_b.\n",
    "    alpha_a / alpha_b: how strongly we map data to semitone 
offset.\n",
    "    \"\"\"\n",
    "\n",
    "    # times at which Stream A triggers\n",
    "    a_times = np.linspace(0, measure_duration*(subdiv_a-
1)/subdiv_a, subdiv_a)\n",
    "    # times at which Stream B triggers\n",
    "    b_times = np.linspace(0, measure_duration*(subdiv_b-
1)/subdiv_b, subdiv_b)\n",
    "\n",
    "    # If no data arrays are given, default to zeros.\n",
```

```
    "    if data_a is None:\n",
    "        data_a = np.zeros(subdiv_a)\n",
    "    if data_b is None:\n",
    "        data_b = np.zeros(subdiv_b)\n",
    "    data_a = data_a[:subdiv_a]\n",
    "    data_b = data_b[:subdiv_b]\n",
    "\n",
    "    # Build an event list: (time, 'A'/'B', freq)\n",
    "    events = []\n",
    "    for i, t_a in enumerate(a_times):\n",
    "        pitch_offset = alpha_a * data_a[i]\n",
    "        final_freq = freq_a * (2.0 ** (pitch_offset/12.0))\n",
    "        events.append((t_a, 'A', final_freq))\n",
    "\n",
    "    for j, t_b in enumerate(b_times):\n",
    "        pitch_offset = alpha_b * data_b[j]\n",
    "        final_freq = freq_b * (2.0 ** (pitch_offset/12.0))\n",
    "        events.append((t_b, 'B', final_freq))\n",
    "\n",
    "    events.sort(key=lambda e: e[0])\n",
    "\n",
    "    # beep_dur is how long each beep lasts\n",
    "    beep_dur = measure_duration / (max(subdiv_a,
subdiv_b)*2)\n",
    "\n",
    "    last_event_time = 0.0\n",
    "    for (t_event, stream_id, freq) in events:\n",
    "        wait_time = t_event - last_event_time\n",
    "        if wait_time > 0:\n",
```

```
    "            sd.sleep(int(wait_time * 1000))\n",
    "        last_event_time = t_event\n",
    "\n",
    "        # generate a short beep\n",
    "        t = np.linspace(0, beep_dur, int(samplerate*beep_dur), endpoint=False)\n",
    "        wave = 0.3 * np.sin(2.0*np.pi*freq*t)\n",
    "\n",
    "        sd.play(wave, samplerate=samplerate)\n",
    "        sd.wait()\n",
    "\n",
    "    # end of measure: if there's leftover time\n",
    "    leftover = measure_duration - last_event_time\n",
    "    if leftover > 0:\n",
    "        sd.sleep(int(leftover*1000))\n",
    "    print(\"Finished one measure of polyrhythm.\")\n",
    "    \n",
    "print(\"play_polyrhythm function is now revised.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "216f5aa5",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
```

```
    "## Single Measure Test\n",
    "Here, we do a single measure with 3:4 subdivisions at measure=2.0s.  Frequencies: A=440, B=220.  No pitch offset."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "a4ae7cf0",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Finished one measure of polyrhythm.\n"
     ]
    }
   ],
   "source": [
    "play_polyrhythm(\n",
    "    measure_duration=2.0,\n",
    "    subdiv_a=3,\n",
    "    subdiv_b=4,\n",
    "    freq_a=440.0,\n",
```

"   freq_b=220.0,\n",
"   data_a=None,\n",
"   data_b=None,\n",
"   alpha_a=0.0,\n",
"   alpha_b=0.0\n",
")"
]
},
{
"cell_type": "markdown",
"id": "a02c35ce",
"metadata": {
"pycharm": {
"name": "#%% md\n"
}
},
"source": [
"## Interactive Sliders\n",
"Let's let you vary measure duration, base frequencies, and the alpha offsets (pitch offsets from data). We'll use small arrays for `data_a` and `data_b` so each sub-trigger can differ in pitch if `alpha_` is nonzero."
]
},
{
"cell_type": "code",
"execution_count": 4,
"id": "1e33b54b",
"metadata": {
"pycharm": {

```
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "59fe91b1e46b4a23bd8a62f735f470c2",
       "version_major": 2,
       "version_minor": 0
      },
      "text/plain": [
       "interactive(children=(FloatSlider(value=2.0, description='measure_duration', max=5.0, min=1.0, step=0.5), IntS…"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "@widgets.interact(\n",
    "    measure_duration=(1.0, 5.0, 0.5),\n",
    "    freq_a=(110, 660, 10),\n",
    "    freq_b=(110, 660, 10),\n",
    "    alpha_a=(0.0, 12.0, 1.0),\n",
    "    alpha_b=(0.0, 12.0, 1.0)\n",
    ")\n",
    "def interactive_polyrhythm(\n",
```

```
    "    measure_duration=2.0,\n",
    "    freq_a=440.0,\n",
    "    freq_b=220.0,\n",
    "    alpha_a=0.0,\n",
    "    alpha_b=0.0\n",
    "):\n",
    "    data_a = np.array([0, 2, 5])  # for the 3-subdiv\n",
    "    data_b = np.array([0, 3, 6, 9])  # for the 4-subdiv\n",
    "\n",
    "    print(\"3:4 polyrhythm => measure=\", measure_duration,\n",
    "          \"freq_a=\", freq_a, \"freq_b=\", freq_b,\n",
    "          \"alpha_a=\", alpha_a, \"alpha_b=\", alpha_b)\n",
    "\n",
    "    play_polyrhythm(\n",
    "        measure_duration=measure_duration,\n",
    "        subdiv_a=3,\n",
    "        subdiv_b=4,\n",
    "        freq_a=freq_a,\n",
    "        freq_b=freq_b,\n",
    "        data_a=data_a,\n",
    "        data_b=data_b,\n",
    "        alpha_a=alpha_a,\n",
    "        alpha_b=alpha_b\n",
    "    )\n",
    "    print(\"Done interactive polyrhythm.\")"
   ]
  },
  {
   "cell_type": "code",
```

```
   "execution_count": null,
   "id": "1d8620f9-a15b-4829-8958-e8def56c671d",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.21"
  }
```

```
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

Chapter 7: Emotional Shading (Optional)

Preliminary Concepts

February 9, 2025

7.1 Introduction

So far, we have focused on "technical" musical parameters: pitch (via a helical axis), timbre (brightness or harmonics), and even rhythm/polyrhythms. Chapter 7 explores an **emotional** or **affective** dimension, often referred to as **emotional shading**.

7.2 Linking Music and Emotion

Historical Attempts Composers and theorists (e.g., the Baroque *Doctrine of Affections*, or the *Romantic era* composers) have long believed that certain **keys**, **chords**, or **intervals** evoke specific moods (e.g., major chords for "happy," minor chords for "sad").

Modern Research Studies in *music psychology* or *affective computing* often associate *valence* (positive/negative) and *arousal* (high/low energy) with different musical features (tempo, mode, timbre).

7.3 Simple Approaches to Emotional Shading

Major vs. Minor Scales

A very common quick approach is:

- **Positive Mood → Major Scale** (e.g. C major).
- **Negative Mood → Minor Scale** (e.g. C natural minor).

This is simplistic, but can have a noticeable emotional coloring.

Dynamics or Tempo

Alternatively, we might increase volume or tempo for high arousal, and decrease them for low arousal, giving a sense of excitement vs. calm.

7.4 Potential Pitfalls

- **Subjectivity:** Not all listeners interpret major/minor as purely happy/sad. Cultural background and personal preference matter.
- **Over-Simplification:** True emotion in music can involve complex chord progressions, tension/resolution patterns, etc.

7.5 Conclusion

Emotional shading is a powerful but subjective extension to data sonification. Even a small shift—like toggling between major and minor or adjusting a timbre parameter to evoke brightness/darkness—can significantly alter the **emotional perception** of the same data. In the expanded discussion, we'll introduce practical ways to incorporate a *mood axis* or *valence parameter* into the Helical Sonification System (HSS).

Next: We explore deeper theoretical and practical considerations in the ***Expanded Discussion*** *for Chapter 7.*

Chapter 7: Emotional Shading (Optional)

Expanded Discussion

February 9, 2025

7.1 Theoretical Foundations of Music-Emotion Links

Hevner's Adjective Circle (1930s) A pioneering attempt to categorize musical emotion with clusters of adjectives (happy, sad, dreamy, vigorous, etc.), showing partial agreement among listeners for certain musical cues (mode, tempo, register).

Russell's Circumplex Model (1980) Defines *valence* (positive to negative) and *arousal* (calm to excited) as orthogonal axes. Music can evoke or occupy a point in this 2D space.

7.2 Minimal Implementation in HSS

Valence Axis

We could map *valence* to a **major** vs. **minor** scale choice. A simple threshold approach:

$$\text{scale} = \begin{cases} \text{majorScale}, & \text{if valence} \geq 0, \\ \text{minorScale}, & \text{if valence} < 0. \end{cases}$$

Alternatively, we can do a smooth blend between major/minor chords if we want intermediate states.

Arousal Axis

- **Tempo Variation**: High arousal $\to$ faster tempo, low arousal $\to$ slower tempo.
- **Amplitude or Dynamics**: High arousal $\to$ louder or more aggressive timbre, low arousal $\to$ softer or gentler timbre.

7.3 Data Sources for Emotion

Explicit vs. Computed Mood One might *directly* ask a user for an "emotion rating" or use an *algorithmic guess* (e.g., sentiment analysis from text or physiological signals). The resulting *valence* or *arousal* value is then mapped to scale choice, tempo, etc.

7.4 Potential Drawbacks

- **Cultural Variation**: Major/minor connotations can vary across cultures, and not everyone perceives major as "happy."
- **Overly Simplistic**: True emotional nuance might need advanced chord progressions, melodic lines, orchestration changes, etc.
- **Listener Expectation**: Some might find it too "gimmicky" if the same data toggles from major to minor without deeper harmonic context.

7.5 Example Approaches Beyond Major/Minor

Continuous Mode Blending Some software (e.g., spectral morphing) can morph chords or scale sets gradually as *valence* changes from negative to positive, giving a subtle shift in color.

Dynamic Expression Beyond scale toggles, we might use *legato/staccato* styles, *crescendo/diminuendo*, or *instrument selection* to represent emotional states.

7.6 Conclusion

Emotional shading is an *optional* but compelling layer for sonification, allowing data to be heard through a lens of "musical feeling." In the accompanying notebook, we show a straightforward method: toggling a major vs. minor scale (valence) and adjusting tempo or amplitude (arousal). Experiment with these ideas to see if they *enhance* the interpretability or expressiveness of your data-based music.

Chapter 7: Emotional Shading and Psychoacoustic Triggers of Fear, Tension, and Excitement

February 9, 2025

7.1 Introduction

In previous chapters, we established a **Helical Sonification System (HSS)** and explored how pitch, timbre, rhythm, and polyrhythms can convey data in a *musically coherent* way. Now, we pivot to what might be the most intriguing domain of all: **emotional shading** and the deliberate evocation of *fear*, *tension*, *excitement*, or *sadness* via sound design. This chapter synthesizes research from *psychoacoustics*, *music psychology*, and *film/game sound design*, revealing the deep biological roots behind certain acoustic cues that reliably trigger emotional and physiological responses.

Key Themes in This Chapter:

- **Psychoacoustic triggers** of fear and tension (e.g., roughness, rapid pitch glides, dissonance).
- **Biological underpinnings**—why certain modulations or intervals mimic animal distress calls and startle our primal threat detection circuits.
- **Tempo, heart rate, and arousal**: how beats per minute can "rev up" or calm down a listener's sympathetic nervous system.
- **Major vs. minor and beyond**: moving past the simplistic "happy vs. sad" mode equation to see how dissonance, chord progressions, and context shape emotional coloring.
- **Case studies** from film (Daleks, Jaws, Psycho) and **video games** (Silent Hill, Resident Evil, DOOM), illustrating how real sound designers exploit psychoacoustic insights.
- **Theoretical frameworks**: e.g., *BRECVEMA* and *GEMS* for mapping musical cues to emotional mechanisms.

We will see that "emotional shading" is far from a simple matter of "major = happy, minor = sad." Indeed, it encompasses **roughness**, **nonlinear noise**, **rising pitch glides**,

rapid tempos that exceed one's resting heart rate, **dissonant intervals**, and **culturally conditioned** associations. Whether you want to inspire a mild sense of unease or full-blown terror, these psychoacoustic tools lie at your disposal.

7.2 Psychoacoustic Elements Triggering Fear and Tension

7.2.1 Roughness and Nonlinear Noise

Animal Alarm Calls and Nonlinearities Research in psychoacoustics indicates that **rough, noisy, inharmonic** sounds instinctively trigger fear, partly because they mimic animal alarm or distress calls. When waveforms become chaotic ("nonlinear"), as in a scream, the human auditory system tends to flag them as threatening.

Studies (Blumstein *et al.*, 2010) found that horror film soundtracks systematically incorporate dissonant noise bursts and rough modulations. Our brains respond more strongly to these "non-musical" sidebands, activating amygdala-based fear circuits. This is an evolutionary adaptation: in nature, "chaotic" vocalizations generally signal high distress.

Amplitude Modulation in the 20–150 Hz Range Roughness arises when two or more partials beat against each other at a *modulation rate* roughly between 20–150 Hz. This rate produces a harsh, grating timbre that the brain associates with screams. *Arnal et al.* (2015) showed that human screams cluster in this roughness band, triggering faster detection and a stronger fear response. Sirens and car alarms exploit similar modulation rates.

7.2.2 Pitch Glides and "Alarm" Signals

Looming Sounds Sudden upward pitch sweeps (e.g., a rising siren) often feel tense or alarming. From a psychoacoustic standpoint, *looming* sounds that increase in frequency or intensity simulate an approaching threat. Neuhoff's research found that listeners overestimate approaching sounds in time, heightening vigilance. Many emergency klaxons alternate pitch up and down, ensuring the ear never fully habituates.

Film and Siren Examples Iconic cases include the shrieking violin glissandos in *Psycho* (imitating a scream) or the cyclical up-down sweeps of an ambulance siren. Both combine pitch glides with roughness, hooking the brainstem reflex before conscious thought. Horror designers love repeating upward slides to keep the listener "on edge"—the impression of a sound rising (or intensifying) warns that something dreadful is imminent.

7.2.3 Amplitude Modulation and Beating Patterns

Fluttering, Jitter, and Screams A small amplitude modulation rate of $\sim$ 4–7 Hz can convey trembling fear (like a shaky voice). As the rate increases to tens of Hz, tremolo turns into harsh roughness. A "fluttery" harshness is central to human screams or certain monster roars (e.g. Dead Space creatures).

Heartbeat Imitation Horror soundtracks occasionally include a thumping that speeds up, mimicking an anxious heartbeat. Human physiology resonates with such rhythmic cues: we subconsciously tense up, possibly raising *our* heart rate in sympathy.

7.3 Musical Tempo, Heart Rate, and Emotional Arousal

Entrainment and BPM Music near or above a listener's resting heart rate ($\sim$ 60–100 BPM) often boosts arousal. Faster beats can accelerate heart rate and breathing (Gomez and Danuser, 2007), whereas slower, "adagio" music relaxes. This phenomenon underpins why action scores or chase themes frequently range from 120 to 150 BPM.

Faster Tempo = Higher Arousal Studies confirm that even a "sad-sounding" minor piece, if played with a very brisk tempo, can shift the overall emotional impression from melancholy to excited. Similarly, a happy major melody played at a *very slow* tempo can feel somber or solemn.

7.3.1 Examples of Tempo Use in Horror/Action

- **Resident Evil Boss Battles:** Music accelerates from $\sim$ 110 BPM to 130+ during intense fights, raising stress and heart rate.
- **DOOM (2016):** transitions from exploration (100 BPM) to heavy combat sections (160+ BPM) for "adrenaline pumping" effect (Mick Gordon's adaptive score).
- **Dead Space:** breathing, heart rate, and soundtrack tempo can all sync, ratcheting up together as panic intensifies.

7.4 Going Beyond "Major = Happy, Minor = Sad"

7.4.1 Cultural Conditioning and Mode Perception

Recent cross-cultural experiments (Smit *et al.*, 2022) show that "major/happy vs. minor/sad" is *not* universal. Rather, it's culturally learned in Western contexts. Listeners from remote areas without Western musical exposure do not consistently map major to happy. Still, among Western-trained ears, major is often perceived as brighter or more uplifting, minor as darker or more introspective.

7.4.2 Dissonance, Chord Progressions, and "Scary Intervals"

Semitone Clashes, Tritones, Tone Clusters Certain intervals (minor seconds, tritones) yield strong dissonance, which correlates with tension or fear. Many horror soundtracks rely on cluster chords—*closely spaced* notes that produce beating or inharmonic partials—resulting in an unsettling, harsh effect (e.g., *Psycho* strings, *Insidious* violin clusters).

The "Devil's Interval" (Tritone) This ΔF is exactly halfway through the octave, creating a sense of unresolved tension. It's so historically reviled as "the devil in music" that medieval theory forbade it. Horror composers exploit that sense of disorientation (e.g. *Halloween* theme has ample use of ambiguous intervals, evoking dread).

Case Study: *Jaws* Motif Just two notes a semitone apart (E–F) hammered in a repeating, accelerating pattern forms one of the most legendary fear triggers in film scoring. The minor second interval is among the most dissonant intervals in Western ears, plus the motif's rhythmic acceleration mimics a heartbeat speeding up.

7.5 Fear and Tension in Video Games: Additional Examples

While film scores are well-studied, **video games** often push these psychoacoustic tricks even further—thanks to interactivity and adaptive music engines.

7.5.1 Silent Hill Series (Akira Yamaoka)

- **Industrial Noise, Metallic Scrapes**: High-pitched sine waves, chaotic distortions, air raid sirens—direct parallels to real alarms that trigger primal alarm reflexes.
- **Low-Frequency Drones** with 60–90 BPM pulses occasionally modulate to match or *slightly exceed* the player's heart rate, fueling subconscious anxiety.

7.5.2 Resident Evil Series

- **Save Room Themes**: Contrasting slow, minor-key piano (to calm after high tension).
- **Boss Battles**: Rapid tempo escalation and dissonant chord progressions (loaded with minor seconds/tritones). Jump-scare stingers (harsh orchestral chords) rely on the brainstem reflex.

7.5.3 Dead Space (2008) *A.L.I.V.E. System*

- Dynamically adjusts tempo, dissonance, and volume based on the player's own heart rate/oxygen levels.
- Eerie, high-frequency glissandos akin to screams plus sub-bass throbs cause a constant sense of dread.
- 80–120 BPM pulses track in-game breathing and stress, leveraging *entrainment* to raise tension.

7.5.4 DOOM (2016) – Mick Gordon's Heavy Metal Score

- Utilizes "choking" distortion and amplitude modulation in the 15–30 Hz band for a rough, "mechanical aggression" vibe.
- Combat sections jump from ~ 100 BPM to 160+, pushing arousal to the extreme—mirroring the frantic demon battles.

7.6 Practical Psychoacoustic Ranges for Horror & Action

Modulation Rates (Fear/Distress)

- **20–150 Hz amplitude modulation** (roughness band) → triggers amygdala/fear (like screams, sirens).
- **30–40 Hz frequency modulation (growling)** → used for monstrous roars or extreme distortion.
- **2–8 Hz amplitude tremolo** → trembling effect, ghostly whispers (Silent Hill).

BPM Ranges for Emotional Impact

- $\sim$ 50–80 BPM: suspense, slow creeping dread.
- $\sim$ 80–110 BPM: building anxiety (Resident Evil's tension loops).
- $\sim$ 120–160 BPM: action, chase sequences, heightened adrenaline (DOOM combat).
- $>$ 160 BPM with dissonance: chaotic panic (extreme boss battles).

Dissonant Intervals and Effects

- **Minor 2nd (semitone)**: high tension (Jaws).
- **Tritone (aug. 4 / dim. 5)**: classic "Devil's Interval," strongly uneasy.
- **Minor 9th (C–Db up an octave)**: extreme dissonance (screechy violin clusters).

7.7 Frameworks for Mapping Music and Emotion

7.7.1 Juslin's BRECVEMA Model

Patrik Juslin's eight mechanisms explain how music induces emotion: **Brain stem reflex**, **Rhythmic entrainment**, **Evaluative conditioning**, **Contagion**, **Visual imagery**, **Episodic memory**, **Musical expectancy**, and **Aesthetic judgment**.

- *Brain stem reflex*: sudden loud/dissonant chord triggers a reflexive startle (horror jump-scares).
- *Rhythmic entrainment*: fast BPM raises heart rate, fueling excitement/fear.
- *Evaluative conditioning*: repeated association of a motif (e.g. Jaws theme) with danger trains us to feel anxious whenever we hear it.

7.7.2 The Geneva Emotional Music Scale (GEMS)

Zentner's GEMS identifies nine music-specific emotion clusters (e.g., *Tension, Wonder, Transcendence*). "Tension" is especially relevant for horror or action scores, capturing uneasy excitement distinct from everyday fear. GEMS acknowledges that we may feel "awe" or "thrill" even in a scary context—music can blur negative and positive affect into complex emotional states.

7.8 Case Studies, Summarizing Insights

7.8.1 Dalek Voices in *Doctor Who*

An iconic sci-fi instance: the ring-modulated "EX-TER-MI-NATE!" Dalek cry is harsh, monotonic, and inhuman. This droning timbre and amplitude sidebands (from a $\sim 30\,\text{Hz}$ carrier) create roughness akin to an alarm call.

7.8.2 *Jaws* Two-Note Motif

Two notes a semitone apart, hammered with increasing speed, represent unstoppable menace. The minor second interval is inherently dissonant; the rhythmic acceleration mimics a rising heartbeat, hooking both *brain stem reflex* and *entrainment*.

7.8.3 *Psycho* Shower Scene Strings

High-pitched violin clusters (close intervals) produce screeching roughness approximating a scream. Sudden, repeated bursts forcibly jolt the listener, epitomizing the "nonlinear call" principle in horror.

7.8.4 Video Game Horror: *Silent Hill*, *Resident Evil*, *Dead Space*

- **Silent Hill**: industrial siren textures, unpredictable noise bursts, mild heartbeat pulses.
- **Resident Evil**: dynamic shift from calm save rooms (slow minor piano) to boss fights with dissonant progression and fast tempo stingers.
- **Dead Space**: adaptive music matches player vitals, plus ear-piercing glissandos and sub-bass throbs for dread.

7.9 Conclusion: Designing Emotional Shading in Sonification

For those applying *Helical Sonification* or similar data-to-sound mapping, these psychoacoustic and emotional insights provide a **powerful extension**. By carefully layering:

- **Pitch glides or dissonance** to convey tension, shock,
- **Rough timbres** or 20–150 Hz amplitude modulation to elicit fear or attention,
- **Tempo and rhythms** that align with or exceed resting heart rate,
- **Mode or chord progressions** that avoid or embrace resolution,
- **Cultural associations** (e.g., major/minor in Western contexts) or repeated "leitmotifs" that condition the listener to expect dread,

we can shape data sonifications that do more than just illustrate trends in a cold, neutral sense. We can evoke real excitement or tension, tapping directly into the listener's primal reflexes and emotional conditioning. This might be valuable for *immersive experiences* (e.g. educational VR games about natural disasters), *medical training simulations* (where stress is part of realism), or simply artistic *data-driven horror installations*.

Key Takeaways :

1. Human fear and tension responses to sound are deeply rooted in *roughness*, *rising pitch*, and *loud/dissonant bursts*.
2. Tempo can literally entrain one's heart rate, making BPM a powerful lever for excitement or relaxation.
3. Cultural factors, learned associations, and chordal language (major vs. minor) color these primal responses, but do not wholly override them.
4. Filmmakers and game designers systematically exploit these psychoacoustic triggers (sirens, glissandos, clusters, sub-bass throbs) to intensify fear or excitement.
5. The *BRECVEMA* framework clarifies *how* these cues induce emotion (brain stem reflex, entrainment, etc.), while the *GEMS* scale clarifies *which emotions* they evoke (tension, awe, sadness, etc.).

All told, emotional shading is arguably the most dramatic dimension we can add to a sonification system—one that can make data "feel alive" or frightening in ways beyond pure intellectual comprehension.

References: (Include references from Blumstein *et al.*, Arnal *et al.*, Gomez & Danuser, Smit *et al.*, etc. as needed, per your preferred citation style, along with the additional references on video games and psychoacoustic frequency ranges.)

Chapter7_Emotion_Notebook.ipynb

```
#%% md
# Chapter 7: Emotional Shading Notebook

This notebook shows a minimal approach:
- We interpret `valence` (negative/positive mood) to choose **major** or **minor** scale.
- We interpret `arousal` (0..something) to set **note duration** or **tempo**.

Then we generate a small data set, map it to pitch, and play them. We let you adjust sliders for
valence/arousal in real-time.
#%%
import numpy as np
import sounddevice as sd
import ipywidgets as widgets
import matplotlib.pyplot as plt
from IPython.display import display

print("Chapter 7 Emotional Shading code imports done.")
#%% md
## Snap & Scale Logic
We reuse a snap-to-scale approach, define major vs. minor scales, and choose which to use based on
`valence`.
#%%
def snap_to_scale(theta_value, scale_set, semitones_per_octave=12):
    octave_int = int(theta_value // semitones_per_octave)
    remainder = theta_value - octave_int*semitones_per_octave
    best_note = None
    best_diff = 9999
    for note in scale_set:
        diff = abs(note - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = note
    snapped_value = octave_int * semitones_per_octave + best_note
    return snapped_value

def semitone_to_freq(base_freq, semitone_offset):
    return base_freq * (2.0 ** (semitone_offset / 12.0))

# Define major and minor scale sets in semitones from a root
major_scale = [0, 2, 4, 5, 7, 9, 11]
minor_scale = [0, 2, 3, 5, 7, 8, 10]

print("snap_to_scale + scale sets ready.")
#%% md
## Emotional Mapping
- If `valence >= 0`, use `major_scale`.
- Otherwise, use `minor_scale`.
- If `arousal` is bigger, we do shorter note duration (faster tempo). If `arousal` is 0, we do
longer notes (slow tempo).
#%%
@widgets.interact(valence=(-1.0,1.0,0.1), arousal=(0.0,5.0,0.5), base_freq=(110,440,10))
def emotional_sonification(valence=0.0, arousal=2.0, base_freq=220):
    """
    - If valence >= 0, pick major scale; else minor.
    - Arousal =>  (0 => slow) up to (5 => quite fast)
    """
    scale = major_scale if valence >= 0 else minor_scale
    # We'll generate a small data set
    data = np.linspace(0, 5, 8)  # 8 points
    alpha = 4.0  # pitch scale factor

    # snap each data point
    pitch_list = []
    for val in data:
        raw_semitone = alpha * val
        snapped = snap_to_scale(raw_semitone, scale)
        pitch_list.append(snapped)

    # note duration depends on arousal => more arousal => shorter
    # let's define note_dur = 0.6 - 0.1*arousal, clamped to min 0.1
    note_dur = max(0.6 - 0.1*arousal, 0.1)

    samplerate = 44100
    for ps in pitch_list:
        freq = semitone_to_freq(base_freq, ps)
        t = np.linspace(0, note_dur, int(samplerate*note_dur), endpoint=False)
```

```
        wave = 0.3 * np.sin(2.0*np.pi*freq*t)
        sd.play(wave, samplerate=samplerate)
        sd.wait()

    # quick visualization
    plt.figure(figsize=(6,3))
    plt.plot(data, 'bo--', label='Data')
    plt.plot(pitch_list, 'ro-', label='PitchSemitones')
    plt.title(f"valence={valence}, arousal={arousal}, scale={'Major' if valence>=0 else 'Minor'}\n"
\
              f"note_dur={note_dur:.2f} s, base_freq={base_freq}")
    plt.legend()
    plt.show()

    print(f"Used {'Major' if valence>=0 else 'Minor'} Scale, note_dur={note_dur:.2f}s")
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 7: Emotional Shading Notebook\n",
    "\n",
    "This notebook shows a minimal approach:\n",
    "- We interpret `valence` (negative/positive mood) to choose
**major** or **minor** scale.\n",
    "- We interpret `arousal` (0..something) to set **note
duration** or **tempo**.\n",
    "\n",
    "Then we generate a small data set, map it to pitch, and play
them. We let you adjust sliders for valence/arousal in real-time."
   ]
  },
  {
   "cell_type": "code",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
```

```
   "source": [
    "import numpy as np\n",
    "import sounddevice as sd\n",
    "import ipywidgets as widgets\n",
    "import matplotlib.pyplot as plt\n",
    "from IPython.display import display\n",
    "\n",
    "print(\"Chapter 7 Emotional Shading code imports done.\")"
   ],
   "execution_count": null
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Snap & Scale Logic\n",
    "We reuse a snap-to-scale approach, define major vs. minor
scales, and choose which to use based on `valence`."
   ]
  },
  {
   "cell_type": "code",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
```

```
    }
   },
   "outputs": [],
   "source": [
    "def snap_to_scale(theta_value, scale_set,
semitones_per_octave=12):\n",
    "    octave_int = int(theta_value // semitones_per_octave)\n",
    "    remainder = theta_value -
octave_int*semitones_per_octave\n",
    "    best_note = None\n",
    "    best_diff = 9999\n",
    "    for note in scale_set:\n",
    "        diff = abs(note - remainder)\n",
    "        if diff < best_diff:\n",
    "            best_diff = diff\n",
    "            best_note = note\n",
    "    snapped_value = octave_int * semitones_per_octave +
best_note\n",
    "    return snapped_value\n",
    "\n",
    "def semitone_to_freq(base_freq, semitone_offset):\n",
    "    return base_freq * (2.0 ** (semitone_offset / 12.0))\n",
    "\n",
    "# Define major and minor scale sets in semitones from a
root\n",
    "major_scale = [0, 2, 4, 5, 7, 9, 11]\n",
    "minor_scale = [0, 2, 3, 5, 7, 8, 10]\n",
    "\n",
    "print(\"snap_to_scale + scale sets ready.\")"
   ],
```

```
   "execution_count": null
  },
  {
   "cell_type": "markdown",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Emotional Mapping\n",
    "- If `valence >= 0`, use `major_scale`.\n",
    "- Otherwise, use `minor_scale`.\n",
    "- If `arousal` is bigger, we do shorter note duration (faster tempo). If `arousal` is 0, we do longer notes (slow tempo)."
   ]
  },
  {
   "cell_type": "code",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
   "source": [
    "@widgets.interact(valence=(-1.0,1.0,0.1), arousal=(0.0,5.0,0.5), base_freq=(110,440,10))\n",
    "def emotional_sonification(valence=0.0, arousal=2.0, base_freq=220):\n",
```

```
    "    \"\"\"\n",
    "    - If valence >= 0, pick major scale; else minor.\n",
    "    - Arousal =>  (0 => slow) up to (5 => quite fast)\n",
    "    \"\"\"\n",
    "    scale = major_scale if valence >= 0 else minor_scale\n",
    "    # We'll generate a small data set\n",
    "    data = np.linspace(0, 5, 8)  # 8 points\n",
    "    alpha = 4.0  # pitch scale factor\n",
    "\n",
    "    # snap each data point\n",
    "    pitch_list = []\n",
    "    for val in data:\n",
    "        raw_semitone = alpha * val\n",
    "        snapped = snap_to_scale(raw_semitone, scale)\n",
    "        pitch_list.append(snapped)\n",
    "\n",
    "    # note duration depends on arousal => more arousal =>
shorter\n",
    "    # let's define note_dur = 0.6 - 0.1*arousal, clamped to min
0.1\n",
    "    note_dur = max(0.6 - 0.1*arousal, 0.1)\n",
    "\n",
    "    samplerate = 44100\n",
    "    for ps in pitch_list:\n",
    "        freq = semitone_to_freq(base_freq, ps)\n",
    "        t = np.linspace(0, note_dur, int(samplerate*note_dur),
endpoint=False)\n",
    "        wave = 0.3 * np.sin(2.0*np.pi*freq*t)\n",
    "        sd.play(wave, samplerate=samplerate)\n",
    "        sd.wait()\n",
```

```
    "\n",
    "    # quick visualization\n",
    "    plt.figure(figsize=(6,3))\n",
    "    plt.plot(data, 'bo--', label='Data')\n",
    "    plt.plot(pitch_list, 'ro-', label='PitchSemitones')\n",
    "    plt.title(f\"valence={valence}, arousal={arousal}, 
scale={'Major' if valence>=0 else 'Minor'}\\n\" \\\n",
    "              f\"note_dur={note_dur:.2f} s, 
base_freq={base_freq}\")\n",
    "    plt.legend()\n",
    "    plt.show()\n",
    "\n",
    "    print(f\"Used {'Major' if valence>=0 else 'Minor'} Scale, 
note_dur={note_dur:.2f}s\")"
   ],
   "execution_count": null
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython"
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
```

```
   "name": "python"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

Chapter 8: Advanced Topics and Scaling Up

Preliminary Concepts

February 9, 2025

8.1 Introduction

So far, we have a robust **Helical Sonification System (HSS)** that handles:

- Pitch (via a helical or spiral mapping),
- Timbre (one-dimensional brightness or harmonic content),
- Rhythm and Polyrhythms (time structuring),
- Emotional Shading (fear, excitement, etc. in Chapter 7).

In Chapter 8, we move beyond "single-stream or small-scale" sonifications and discuss **advanced applications** and **scaling up**. This involves:

- Multi-voice or multi-layered sonification (assigning multiple instruments or channels to different data dimensions),
- Handling large or high-dimensional data (e.g., time-series with numerous variables),
- Potential for real-time or interactive setups, such as **VR** or **data gloves** that let users modulate parameters on the fly.

Goals in This Chapter

1. Introduce multi-voice architecture: how to split data into separate parts (pitch, timbre, second instrument, polyrhythm, etc.).
2. Outline strategies for large data sets: chunking, layering, or data reduction (PCA or clustering).
3. Touch on real-time implementations: how to handle continuous user input or live data updates.

These concepts tie together everything we've learned, aiming for truly immersive and powerful sonification experiences.
We'll keep the overview concise here; the next "Expanded Discussion" will cover each point with examples, references, and potential pitfalls.

Chapter 8: Advanced Topics and Scaling Up

Expanded Discussion

February 9, 2025

8.1 Multi-Voice Sonification

8.1.1 Why Multiple Voices?

When data sets grow more complex (e.g., multiple variables, high-dimensional time-series), encoding everything in a *single pitch-timbre axis* quickly becomes overwhelming. Splitting the data across two or more "voices" or instruments can:

- Keep each channel simpler, reducing "sonic overload,"
- Let listeners hear separate streams as distinct musical lines,
- Allow more nuanced expression or interplay (like call-and-response or polyrhythms).

8.1.2 Assigning Variables to Voices

Suppose you have four variables (v_1, v_2, v_3, v_4). You might:

- Map v_1 to pitch + timbre in **Voice A** (like we did in earlier examples),
- Map v_2 to pitch + brightness in **Voice B**, possibly an octave higher,
- Keep v_3, v_4 for polyrhythmic triggers or amplitude modulation in those voices.

Alternatively, *Voice A* runs a 3-subdivision pattern, *Voice B* does 4-subdivision, yielding a polyrhythm plus melodic interplay.

8.1.3 Pitfalls and Tips

- **Cognitive Load**: Each additional voice is another stream the listener has to disentangle. Limit to 2–4 distinct voices unless you have very advanced users or use simpler pitch mappings.
- **Register Separation**: Placing voices in distinct pitch ranges (e.g., *Voice A* in a lower register, *Voice B* higher) helps the ear separate them.
- **Instrument/Timbre Choice**: E.g., one voice with a string timbre, another with a bell-like or brass timbre, to avoid blending confusion.

8.2 Approaches for Large Data Sets

8.2.1 Chunking or Time-Slicing

If you have a time-series of thousands of points, playing them all in quick succession can create a "vacuum cleaner" sound. Instead:

- Break data into smaller *windows* or *chunks*,
- Summarize each chunk (mean, median, or max-min range) before mapping to pitch,
- Possibly convert a big data set into multiple shorter "movements" in the sonification.

8.2.2 Dimensionality Reduction (PCA, Clustering)

For high-dimensional data (e.g. 10+ variables):

- **Principal Component Analysis (PCA)** can compress the data into a few principal axes, each mapped to a different sonic dimension (pitch, timbre, polyrhythm).
- **Clustering** can group data points into categories, each category assigned to a different instrument or chord.

This ensures you *focus* on core data patterns instead of trying to blindly sonify each dimension.

8.2.3 Data Overlays and Layering

Another tactic is layering multiple "sonic textures" on top:

- **Base Layer**: fundamental pitch line representing the main trend,
- **Overlay Layer**: short rhythmic motifs when an event or threshold is crossed,
- **Ambient Drone**: representing global average or standard deviation as a slowly shifting chord/timbre.

This "soundscape" approach can be more immersive but requires careful volume balancing.

8.3 Real-Time and Interactive Implementations

8.3.1 Why Real-Time?

Many advanced use-cases want immediate feedback:

- Scientific labs monitoring live sensor data (e.g. *EEG* or *seismographs*),
- Interactive exhibits where users "scrub" through data or manipulate parameters,
- Virtual/augmented reality experiences letting participants "move" in a data space and hear changes on the fly.

8.3.2 Technical Considerations

- **Latency**: Generating and playing sounds with minimal delay (avoid your engine taking 500 ms or more to respond).
- **Buffering Strategy**: Possibly use small audio buffers for a near-immediate reaction, at the cost of more CPU usage.
- **Control Interfaces**: e.g. *MIDI controllers*, *Leap Motion* sensors, VR hand tracking, etc. Provide intuitive parameter knobs or sliders for pitch scale, timbre brightness, rhythmic density, etc.

8.3.3 Example: VR "Data Playground"

One might build a VR environment where each axis in 3D space (or 4D, if you incorporate polyrhythmic time) corresponds to some data dimension. As users walk or "fly" through the space, pitch lines, timbres, and rhythms change accordingly. This merges data exploration with spatial movement and immediate sonic feedback, potentially allowing quick pattern recognition.

8.4 Putting It All Together: Potential Pitfalls

- **Over-Complex Sound**: Multi-voice + polyrhythm + real-time manipulations can become a cacophony. Provide user toggles or a "mute voice" feature.
- **Performance Bottlenecks**: Real-time audio with big data. Preprocessing or partial caching might be needed if the data is massive.
- **User Training**: Not everyone can parse multi-layered sonification instantly. Offer layered learning or tutorials in a data exhibit or VR environment.

8.5 Conclusion

Chapter 8 expands our Helical Sonification System to handle **multiple voices**, **large data sets**, and **real-time interactivity**. These techniques:

- allow deeper, more complex data representations,
- enable dynamic, user-driven experiences (in VR or live data scenarios),
- and illustrate how musical sonification can scale from a single melodic line to an entire "soundscape" that conveys many aspects of the data in parallel.

In the next section (code examples), we provide a minimal multi-voice demonstration plus some tips for chunking large data. Real-time interactive examples are also shown to spark ideas about how to incorporate your own data streams or user inputs.

Chapter8_Advanced_Notebook.ipynb

```
#%% md
# Chapter 8: Advanced Topics and Scaling Up (Notebook)

In this notebook, we:
1. Demonstrate **multi-voice** sonification (two separate data arrays, each with pitch + timbre in
different registers).
2. Show a **chunking** approach for large data, summarizing in smaller blocks before mapping.
3. Provide an **interactive** cell using sliders for user adjustments, simulating partial real-time
or live re-sonification.
#%%
import numpy as np
import sounddevice as sd
import ipywidgets as widgets
import matplotlib.pyplot as plt
print("Chapter 8 advanced notebook imports loaded.")
#%% md
## 1) Multi-Voice Sonification
We'll define two functions: one for voice A, one for voice B. Each voice gets data mapped to pitch
& timbre (like in Chapter 3-4). We'll keep them in different pitch registers or timbre offsets so
they don't overlap too much.
#%%
# A quick scale snapping approach, reusing code from prior chapters.
def snap_to_scale(theta_value, scale_set=[0,2,4,5,7,9,11], semitones_per_octave=12):
    octave_int = int(theta_value // semitones_per_octave)
    remainder = theta_value - octave_int*semitones_per_octave
    best_note = None
    best_diff = 9999
    for note in scale_set:
        diff = abs(note - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = note
    return octave_int*semitones_per_octave + best_note

def semitone_to_freq(base_freq, semitone_offset):
    return base_freq * (2.0 ** (semitone_offset / 12.0))

def play_two_voice(
    dataA, dataB,
    alphaA=4.0, alphaB=4.0,
    baseA=220.0, baseB=440.0,
    note_dur=0.3, scaleA=[0,2,4,5,7,9,11], scaleB=[0,2,4,5,7,9,11]
):
    """
    dataA -> voice A (pitch/timbre?), dataB -> voice B.
    alphaA/alphaB scale factors for pitch.
    baseA/baseB: base frequencies for each voice's reference.
    We'll just do them in sequence for demonstration.
    """
    samplerate = 44100

    # Snap dataA -> pitchA
    pitchA = []
    for val in dataA:
        raw_semA = alphaA*val
        snappedA = snap_to_scale(raw_semA, scaleA)
        pitchA.append(snappedA)

    # Snap dataB -> pitchB
    pitchB = []
    for val in dataB:
        raw_semB = alphaB*val
        snappedB = snap_to_scale(raw_semB, scaleB)
        pitchB.append(snappedB)

    # We'll alternate playing a note from A, then a note from B, etc.
    # In real multi-voice, we might want concurrency or layering.
    # For simplicity, let's do a quick A->B->A->B...

    max_len = max(len(pitchA), len(pitchB))
    indexA = 0
    indexB = 0

    while indexA < len(pitchA) or indexB < len(pitchB):
        # Play next A note if available
        if indexA < len(pitchA):
```

```
            freqA = semitone_to_freq(baseA, pitchA[indexA])
            waveA = 0.3 * np.sin(2.0*np.pi*freqA * np.linspace(0, note_dur,
int(samplerate*note_dur), endpoint=False))
            sd.play(waveA, samplerate=samplerate)
            sd.wait()
            indexA += 1

        # Then play next B note if available
        if indexB < len(pitchB):
            freqB = semitone_to_freq(baseB, pitchB[indexB])
            waveB = 0.3 * np.sin(2.0*np.pi*freqB * np.linspace(0, note_dur,
int(samplerate*note_dur), endpoint=False))
            sd.play(waveB, samplerate=samplerate)
            sd.wait()
            indexB += 1

    print("Two-voice playback done.")

print("play_two_voice function ready.")
#%% md
### Quick Test
We'll define two small arrays (like `[0,1,2,3]` and `[2,4,5,1]`) and see them played in an A->B->A->B sequence.
#%%
dataA = [0,1,2,3]
dataB = [2,4,5,1]
play_two_voice(dataA, dataB, alphaA=4.0, alphaB=4.0,
               baseA=220.0, baseB=440.0, note_dur=0.3)
#%% md
## 2) Handling Larger Data via Chunking
We'll demonstrate a simple chunking approach: if we have, say, 100 points, we'll divide them into
10 chunks of 10 points each, taking the mean of each chunk to create 10 data points for playback.
#%%
def chunk_data(data, chunk_size=10, method='mean'):
    """
    Splits 'data' into blocks of 'chunk_size' and returns one value per chunk.
    method can be 'mean', 'max', etc.
    """
    out = []
    length = len(data)
    for i in range(0, length, chunk_size):
        block = data[i:i+chunk_size]
        if method=='mean':
            val = np.mean(block)
        elif method=='max':
            val = np.max(block)
        else:
            val = np.mean(block)  # default
        out.append(val)
    return out

print("chunk_data function ready.")
#%% md
### Example: 100 random points in range [0..5], chunked into 10 blocks
#%%
long_dataA = np.random.uniform(0, 5, 100)
long_dataB = np.random.uniform(0, 5, 100)

chunkedA = chunk_data(long_dataA, chunk_size=10, method='mean')
chunkedB = chunk_data(long_dataB, chunk_size=10, method='mean')

print("Chunked A:", chunkedA)
print("Chunked B:", chunkedB)

# Now we can feed these chunked arrays into play_two_voice.
play_two_voice(chunkedA, chunkedB, alphaA=4.0, alphaB=5.0,
               baseA=220.0, baseB=330.0, note_dur=0.3)
print("Done with chunked data playback.")
#%% md
## 3) A Simple Interactive Cell (Parameters for Multi-Voice)
We'll let you adjust:
- `alphaA` (scale factor for dataA pitch)
- `alphaB` (scale factor for dataB pitch)
- `note_dur`
- `method` for chunking (mean or max)
Then we chunk a random dataset each time you change sliders and do a new multi-voice playback.
#%%
from ipywidgets import interact, FloatSlider, Dropdown

@interact(
    alphaA=(1.0, 8.0, 1.0),
```

```
    alphaB=(1.0, 8.0, 1.0),
    note_dur=(0.1, 1.0, 0.1),
    method=Dropdown(options=['mean','max'], value='mean', description='ChunkMethod')
)
def multi_voice_interactive(alphaA=4.0, alphaB=5.0, note_dur=0.3, method='mean'):
    # regenerate random data each time
    bigA = np.random.uniform(0, 5, 60)
    bigB = np.random.uniform(0, 5, 60)
    chunkedA = chunk_data(bigA, chunk_size=10, method=method)
    chunkedB = chunk_data(bigB, chunk_size=10, method=method)

    print("Playing with alphaA=", alphaA, "alphaB=", alphaB,
          "note_dur=", note_dur, "method=", method)

    play_two_voice(
        chunkedA, chunkedB,
        alphaA=alphaA, alphaB=alphaB,
        baseA=220.0, baseB=330.0,
        note_dur=note_dur
    )
    print("Interaction done.")
#%%
```

{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "602d3acf",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 8: Advanced Topics and Scaling Up (Notebook)\n",
    "\n",
    "In this notebook, we:\n",
    "1. Demonstrate **multi-voice** sonification (two separate data arrays, each with pitch + timbre in different registers).\n",
    "2. Show a **chunking** approach for large data, summarizing in smaller blocks before mapping.\n",
    "3. Provide an **interactive** cell using sliders for user adjustments, simulating partial real-time or live re-sonification."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "fbd28462",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }

```
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Chapter 8 advanced notebook imports loaded.\n"
     ]
    }
   ],
   "source": [
    "import numpy as np\n",
    "import sounddevice as sd\n",
    "import ipywidgets as widgets\n",
    "import matplotlib.pyplot as plt\n",
    "print(\"Chapter 8 advanced notebook imports loaded.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "982a2ce8",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## 1) Multi-Voice Sonification\n",
```

```
    "We'll define two functions: one for voice A, one for voice B. Each voice gets data mapped to pitch & timbre (like in Chapter 3-4). We'll keep them in different pitch registers or timbre offsets so they don't overlap too much."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "616be0f2",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "play_two_voice function ready.\n"
     ]
    }
   ],
   "source": [
    "# A quick scale snapping approach, reusing code from prior chapters.\n",
    "def snap_to_scale(theta_value, scale_set=[0,2,4,5,7,9,11], semitones_per_octave=12):\n",
    "    octave_int = int(theta_value // semitones_per_octave)\n",
```

```
    "    remainder = theta_value - 
octave_int*semitones_per_octave\n",
    "    best_note = None\n",
    "    best_diff = 9999\n",
    "    for note in scale_set:\n",
    "        diff = abs(note - remainder)\n",
    "        if diff < best_diff:\n",
    "            best_diff = diff\n",
    "            best_note = note\n",
    "    return octave_int*semitones_per_octave + best_note\n",
    "\n",
    "def semitone_to_freq(base_freq, semitone_offset):\n",
    "    return base_freq * (2.0 ** (semitone_offset / 12.0))\n",
    "\n",
    "def play_two_voice(\n",
    "    dataA, dataB,\n",
    "    alphaA=4.0, alphaB=4.0,\n",
    "    baseA=220.0, baseB=440.0,\n",
    "    note_dur=0.3, scaleA=[0,2,4,5,7,9,11], 
scaleB=[0,2,4,5,7,9,11]\n",
    "):\n",
    "    \"\"\"\n",
    "    dataA -> voice A (pitch/timbre?), dataB -> voice B.\n",
    "    alphaA/alphaB scale factors for pitch.\n",
    "    baseA/baseB: base frequencies for each voice's 
reference.\n",
    "    We'll just do them in sequence for demonstration.\n",
    "    \"\"\"\n",
    "    samplerate = 44100\n",
    "\n",
```

```
    "    # Snap dataA -> pitchA\n",
    "    pitchA = []\n",
    "    for val in dataA:\n",
    "        raw_semA = alphaA*val\n",
    "        snappedA = snap_to_scale(raw_semA, scaleA)\n",
    "        pitchA.append(snappedA)\n",
    "\n",
    "    # Snap dataB -> pitchB\n",
    "    pitchB = []\n",
    "    for val in dataB:\n",
    "        raw_semB = alphaB*val\n",
    "        snappedB = snap_to_scale(raw_semB, scaleB)\n",
    "        pitchB.append(snappedB)\n",
    "\n",
    "    # We'll alternate playing a note from A, then a note from
B, etc.\n",
    "    # In real multi-voice, we might want concurrency or
layering.\n",
    "    # For simplicity, let's do a quick A->B->A->B...\n",
    "\n",
    "    max_len = max(len(pitchA), len(pitchB))\n",
    "    indexA = 0\n",
    "    indexB = 0\n",
    "\n",
    "    while indexA < len(pitchA) or indexB < len(pitchB):\n",
    "        # Play next A note if available\n",
    "        if indexA < len(pitchA):\n",
    "            freqA = semitone_to_freq(baseA, pitchA[indexA])\n",
```

```
    "            waveA = 0.3 * np.sin(2.0*np.pi*freqA *
np.linspace(0, note_dur, int(samplerate*note_dur),
endpoint=False))\n",
    "            sd.play(waveA, samplerate=samplerate)\n",
    "            sd.wait()\n",
    "            indexA += 1\n",
    "\n",
    "        # Then play next B note if available\n",
    "        if indexB < len(pitchB):\n",
    "            freqB = semitone_to_freq(baseB, pitchB[indexB])\n",
    "            waveB = 0.3 * np.sin(2.0*np.pi*freqB *
np.linspace(0, note_dur, int(samplerate*note_dur),
endpoint=False))\n",
    "            sd.play(waveB, samplerate=samplerate)\n",
    "            sd.wait()\n",
    "            indexB += 1\n",
    "\n",
    "    print(\"Two-voice playback done.\")\n",
    "\n",
    "print(\"play_two_voice function ready.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "df23ed66",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
```

```
   "source": [
    "### Quick Test\n",
    "We'll define two small arrays (like `[0,1,2,3]` and `[2,4,5,1]`) and see them played in an A->B->A->B sequence."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "e7ac676f",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Two-voice playback done.\n"
     ]
    }
   ],
   "source": [
    "dataA = [0,1,2,3]\n",
    "dataB = [2,4,5,1]\n",
    "play_two_voice(dataA, dataB, alphaA=4.0, alphaB=4.0,\n",
    "               baseA=220.0, baseB=440.0, note_dur=0.3)"
```

```
   ]
  },
  {
   "cell_type": "markdown",
   "id": "275855c1",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## 2) Handling Larger Data via Chunking\n",
    "We'll demonstrate a simple chunking approach: if we have, say, 100 points, we'll divide them into 10 chunks of 10 points each, taking the mean of each chunk to create 10 data points for playback."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "id": "d48c116c",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
```

```
     "output_type": "stream",
     "text": [
      "chunk_data function ready.\n"
     ]
    }
   ],
   "source": [
    "def chunk_data(data, chunk_size=10, method='mean'):\n",
    "    \"\"\"\n",
    "    Splits 'data' into blocks of 'chunk_size' and returns one value per chunk.\n",
    "    method can be 'mean', 'max', etc.\n",
    "    \"\"\"\n",
    "    out = []\n",
    "    length = len(data)\n",
    "    for i in range(0, length, chunk_size):\n",
    "        block = data[i:i+chunk_size]\n",
    "        if method=='mean':\n",
    "            val = np.mean(block)\n",
    "        elif method=='max':\n",
    "            val = np.max(block)\n",
    "        else:\n",
    "            val = np.mean(block)  # default\n",
    "        out.append(val)\n",
    "    return out\n",
    "\n",
    "print(\"chunk_data function ready.\")"
   ]
  },
```

```
  {
   "cell_type": "markdown",
   "id": "1df6ee81",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "### Example: 100 random points in range [0..5], chunked into 10 blocks"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "id": "30ffb4b7",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Chunked A: [np.float64(2.245984517317331), np.float64(2.368732502938418), np.float64(2.135971935812843), np.float64(2.538067516163033), np.float64(2.872884340723584),
```

```
np.float64(2.873697743251653), np.float64(2.2926306935064984),
np.float64(2.7160638766037684), np.float64(1.9570360440453267),
np.float64(2.513906030574194)]\n",
      "Chunked B: [np.float64(2.6082919781947504),
np.float64(2.2945707638421453), np.float64(2.435395789078543),
np.float64(2.3118541842412097), np.float64(2.3767517449606395),
np.float64(2.258556867660652), np.float64(3.1600880971787007),
np.float64(2.952925482947049), np.float64(2.4580340162059673),
np.float64(3.574137428029742)]\n",
      "Two-voice playback done.\n",
      "Done with chunked data playback.\n"
     ]
    }
   ],
   "source": [
    "long_dataA = np.random.uniform(0, 5, 100)\n",
    "long_dataB = np.random.uniform(0, 5, 100)\n",
    "\n",
    "chunkedA = chunk_data(long_dataA, chunk_size=10,
method='mean')\n",
    "chunkedB = chunk_data(long_dataB, chunk_size=10,
method='mean')\n",
    "\n",
    "print(\"Chunked A:\", chunkedA)\n",
    "print(\"Chunked B:\", chunkedB)\n",
    "\n",
    "# Now we can feed these chunked arrays into play_two_voice.\n",
    "play_two_voice(chunkedA, chunkedB, alphaA=4.0, alphaB=5.0,\n",
    "               baseA=220.0, baseB=330.0, note_dur=0.3)\n",
    "print(\"Done with chunked data playback.\")"
   ]
  },
```

```
  {
   "cell_type": "markdown",
   "id": "4c004e1e",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## 3) A Simple Interactive Cell (Parameters for Multi-Voice)\n",
    "We'll let you adjust:\n",
    "- `alphaA` (scale factor for dataA pitch)\n",
    "- `alphaB` (scale factor for dataB pitch)\n",
    "- `note_dur`\n",
    "- `method` for chunking (mean or max)\n",
    "Then we chunk a random dataset each time you change sliders and do a new multi-voice playback."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "id": "e3c3eb39",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
```

```
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "9fd5a659697345f595d161b555a21b2f",
       "version_major": 2,
       "version_minor": 0
      },
      "text/plain": [
       "interactive(children=(FloatSlider(value=4.0,
description='alphaA', max=8.0, min=1.0, step=1.0), FloatSlider(va…"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "from ipywidgets import interact, FloatSlider, Dropdown\n",
    "\n",
    "@interact(\n",
    "    alphaA=(1.0, 8.0, 1.0),\n",
    "    alphaB=(1.0, 8.0, 1.0),\n",
    "    note_dur=(0.1, 1.0, 0.1),\n",
    "    method=Dropdown(options=['mean','max'], value='mean',
description='ChunkMethod')\n",
    ")\n",
    "def multi_voice_interactive(alphaA=4.0, alphaB=5.0,
note_dur=0.3, method='mean'):\n",
    "    # regenerate random data each time\n",
```

```
    "    bigA = np.random.uniform(0, 5, 60)\n",
    "    bigB = np.random.uniform(0, 5, 60)\n",
    "    chunkedA = chunk_data(bigA, chunk_size=10, 
method=method)\n",
    "    chunkedB = chunk_data(bigB, chunk_size=10, 
method=method)\n",
    "\n",
    "    print(\"Playing with alphaA=\", alphaA, \"alphaB=\", 
alphaB,\n",
    "          \"note_dur=\", note_dur, \"method=\", method)\n",
    "\n",
    "    play_two_voice(\n",
    "        chunkedA, chunkedB,\n",
    "        alphaA=alphaA, alphaB=alphaB,\n",
    "        baseA=220.0, baseB=330.0,\n",
    "        note_dur=note_dur\n",
    "    )\n",
    "    print(\"Interaction done.\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "ca87a1cd-aa9b-47c3-aea1-359cba79bf1e",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
```

```
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.21"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

Chapter 9: Case Studies & Future Directions

Preliminary Concepts

February 9, 2025

9.1 Introduction

Having established a robust **Helical Sonification System (HSS)**, along with emotional shading, multi-voice strategies, and other advanced techniques, we now turn to *practical case studies* and *future directions*. Chapter 9 highlights real-world examples, bridging the gap between theoretical frameworks and concrete implementations.

Case Study Examples

- Well-known physics or mathematical problems (the brachistochrone, planetary orbits, wave equations).
- Statistical data sets (stock market trends, population growth, climate data).
- Complex shapes (fractals, parametric curves) that can benefit from multi-axis or polyrhythmic approaches.

Future Directions

We also outline emerging topics, such as:

- Integration with **machine learning** or **AI-driven** composition,
- **VR/AR** immersive sonification, letting users "walk" through data,
- More advanced emotional or polyrhythmic expansions to handle high-dimensional or time-evolving data.

We keep this preview short—next, we dive deeper into each case and speculate on what the future might hold for data-driven music and interactive sonification.

Chapter 9: Case Studies & Future Directions

Expanded Discussion

February 10, 2025

9.1 Deep-Dive Case Studies

9.1.1 Brachistochrone Problem: Sonifying a Classic Physics Curve

Background The brachistochrone is the curve of quickest descent under uniform gravity—famously solved by Bernoulli, Newton, and Leibniz. It's known to be a cycloid. Even today, it's a shining example of how nontrivial the "fastest path" can be.

Sonification Approach

- **X-axis** (parametric time) $\rightarrow$ note triggers or rhythmic steps,
- **Y-value** of the cycloid $\rightarrow$ pitch, snapped to a musical scale,
- **Optional Timbre** dimension: curvature or slope of the curve can modulate brightness (steeper slope = brighter timbre).

Users can "hear" how the cycloid yields a swifter travel time compared to a simple parabola or other guesses. The pitch changes more swiftly in the cycloid case, reflecting its faster arrival.

9.1.2 Planetary Orbits & Orbital Resonances

In orbital mechanics, planets and moons often form resonance relationships (e.g. a 2:3 ratio in orbital periods). We can assign each body a voice that *pulses* or *triggers* when it completes a fraction of an orbit. The resulting polyrhythmic interplay reveals any near-integer resonances:

- **Voice A** = Jupiter's orbital phase,
- **Voice B** = Saturn's orbital phase,
- When they sync up (once every few years), the pulses align or create a simpler ratio polyrhythm, which is audible as a moment of "consonance."

Such an approach elegantly demonstrates resonance phenomena that might otherwise remain abstract in numeric ephemerides.

9.1.3 Statistical or Real-World Data: Climate Trends, Stocks, *etc.*

Climate Example Take global temperature anomalies over 150 years. Using chunking (Chapter 8) plus helical pitch mapping, we can hear long-term warming trends as a rising pitch baseline, with short-term cyclical variations forming melodic oscillations.

Stock Market Example Mapping daily stock closes or volume to pitch changes or timbre can highlight extreme volatility spikes. Some traders even rely on "ear" recognition of chart patterns, hearing repeated fluctuations or meltdown-like crashes as sudden dissonant bursts or faster polyrhythms.

9.2 Future Directions for Sonification Systems

9.2.1 Machine Learning and AI-Driven Composition

As **ML** and **generative AI** models become more sophisticated, we might see:

- Automated mapping strategies that adapt to user feedback (an AI deciding how best to encode certain variables),
- Real-time generation of *musical themes* for different data clusters,
- Use of large language models to parse *semantic* data relationships, then propose emotive chord progressions or timbral shifts that highlight key patterns.

This synergy could yield extremely adaptive and personalized sonifications.

9.2.2 VR/AR Immersive Data Exploration

- **Spatial Audio**: integrating 3D positioning so different data streams come from distinct directions, or revolve around the user.
- **Gesture Control**: letting the user "grab" a curve and drag it around, hearing pitch/timbre changes in real time, or scrubbing through parametric surfaces.
- **Polyrhythmic Animations**: walking physically in VR, while each footstep triggers data-driven pulses or melodic fragments that correspond to the user's location in the data space.

Such immersive experiences may revolutionize how we "analyze" big data—imagine hearing + seeing a 10D dataset by literally stepping through principal components in VR.

9.2.3 Emotional Axes + Psychophysiological Feedback

Chapter 7 introduced emotional shading. We might take it further:

- **Biofeedback loops**: track the user's heart rate or galvanic skin response, adjusting the sonification's tension/dissonance in real time. A high stress reading might dial back the complexity or volume.
- **Therapeutic or meditative sonification**: using gentle pitch spirals, slow polyrhythms, or major/minor manipulations to help users calm down or reflect on data in a more emotional way.

Such expansions highlight the *affective* dimension as more than just a novelty, becoming a genuine user-centered design approach.

9.3 Potential Pitfalls & Challenges

9.3.1 Overly Complex "Sound Worlds"

A common challenge: as we add more data variables, emotional shading, multi-voice layering, the sonic environment can become dense or chaotic. We risk losing clarity:

- **User Training** is crucial. Provide interactive tutorials or incremental layering.
- **Option to Mute or Isolate** voices. For instance, let the user toggle "timbre voice off" if it's overshadowing pitch voice.

9.3.2 Hardware and Latency Constraints

Real-time VR or interactive setups might demand sub-50 ms latency to feel responsive. Large data sets can hamper performance. Solutions involve:

- Buffering or partial precomputation,
- Using specialized audio frameworks or libraries (Csound, Wwise, FMOD) that handle dynamic music well,
- Possibly offloading heavy data transforms to a GPU or HPC server.

9.4 Conclusion

Chapter 9 demonstrates how the *Helical Sonification System* expands to real-world complexities:

- We can *case-study* classical math/physics (brachistochrone, orbits) or large modern data sets (climate, stocks).

- We can push into *future directions* like AI-based composition, VR-based data exploration, or biofeedback-driven emotional shading.

Hence, the journey from a single pitch function to a *multi-voice, real-time emotional sonification environment* is well within reach. In the following code examples, we'll illustrate at least one or two "case study" prototypes (e.g., brachistochrone, fractal data) and some forward-looking expansions.

chapter9_cases_notebook.ipynb

```
#%% md
# Chapter 9: Case Studies & Future Directions (Notebook)

We'll:
1. Demonstrate a **Brachistochrone** curve example, mapping its (x,y) param to pitch/timbre.
2. Show how to do a small **fractal** or param.
3. Outline a mock "AI approach" concept or a VR concept (we won't do full VR here, but talk about a
function that might generate relevant data).
#%%
import numpy as np
import sounddevice as sd
import ipywidgets as widgets
import matplotlib.pyplot as plt
print("Chapter 9 Case Studies imports loaded.")
#%% md
## 1) Brachistochrone Parametric Sonification
Recall that the brachistochrone is typically parameterized by cycloid equations:
\[
  x(\theta) = r (\theta - \sin\theta),\quad y(\theta) = r (1 - \cos\theta).
\]
We'll pick a range for \(\theta\) (e.g. 0..$2\pi$). Then map $y(\theta)$ to pitch, maybe
$x(\theta)$ to a secondary dimension (like timbre). We'll do a short code snippet to illustrate.
#%%
# Simple scale snapping
def snap_to_scale(value, scale=[0,2,4,5,7,9,11], semis=12):
    octave = int(value // semis)
    rem = value - octave*semis
    best = None
    best_diff = 9999
    for s in scale:
        diff = abs(s - rem)
        if diff<best_diff:
            best_diff = diff
            best = s
    return octave*semis + best

def semitone_to_freq(base, offset):
    return base*(2.0**(offset/12.0))

def brachistochrone_sonify(r=1.0, theta_max=2*np.pi, steps=20, alpha_y=10.0, alpha_x=2.0,
base_freq=220.0, note_dur=0.3):
    """
    We'll param from theta=0..theta_max.
    Map y->pitch, x->some timbre offset (or brightness) for example.
    We'll do a simple approach: we play a note at each step.
    """
    thetas = np.linspace(0, theta_max, steps)
    xvals = r*(thetas - np.sin(thetas))
    yvals = r*(1.0 - np.cos(thetas))

    # We'll map y to pitch.
    # alpha_y scales y.
    # x might scale amplitude of second harmonic.
    samplerate=44100

    for i in range(steps):
        pitch_val = alpha_y*yvals[i]
        snapped = snap_to_scale(pitch_val)
        freq = semitone_to_freq(base_freq, snapped)

        x_timbre = alpha_x*xvals[i]
        # let's interpret x_timbre as amplitude of a second harmonic.
        # negative x won't matter much, we'll just do abs.
        brightness = abs(x_timbre)*0.1
        if brightness>1.0:
            brightness=1.0

        t = np.linspace(0, note_dur, int(samplerate*note_dur), endpoint=False)
        fundamental = 0.3*np.sin(2.0*np.pi*freq*t)
        second_harm = 0.3*brightness*np.sin(2.0*np.pi*(2*freq)*t)
        wave = fundamental+second_harm

        sd.play(wave, samplerate=samplerate)
        sd.wait()

    print("Brachistochrone sonification done.")
```

```
    # Quick plot
    plt.figure(figsize=(5,4))
    plt.plot(xvals, yvals, 'bo-', label='Brachistochrone')
    plt.title("Brachistochrone (r={:.2f})".format(r))
    plt.xlabel("x")
    plt.ylabel("y")
    plt.legend()
    plt.show()

print("brachistochrone_sonify function ready.")
#%% md
### Quick Test with default values
#%%
brachistochrone_sonify(r=1.0, theta_max=2*np.pi, steps=12, alpha_y=10.0, alpha_x=2.0,
base_freq=220.0, note_dur=0.3)
#%% md
## 2) Fractal / Parametric Example
We'll do a small param (like a Lissajous or a logistic map) just to show how you might map 2D data
to pitch & timbre again. We'll keep it short.
#%%
def logistic_map_sonify(r=3.5, x0=0.2, steps=30, alpha=12.0, base_freq=220.0, note_dur=0.2):
    """
    The logistic map: x_{n+1} = r*x_n*(1 - x_n), 0<x<1.
    We'll track x and sonify.
    """
    samplerate=44100
    x = x0
    out = []
    for i in range(steps):
        out.append(x)
        x = r*x*(1-x)

    # Each value in [0..1], we can scale it up.
    for val in out:
        pitch_val = alpha*val  # ~ 0..alpha
        semitones = snap_to_scale(pitch_val)
        freq = semitone_to_freq(base_freq, semitones)

        # simple wave
        t = np.linspace(0, note_dur, int(samplerate*note_dur), endpoint=False)
        wave = 0.3*np.sin(2.0*np.pi*freq*t)
        sd.play(wave, samplerate=samplerate)
        sd.wait()

    print("Logistic map sonification done.")

    # quick plot
    plt.figure(figsize=(5,4))
    plt.plot(out, 'r.-')
    plt.title(f"Logistic map, r={r}")
    plt.xlabel("Step")
    plt.ylabel("x")
    plt.show()

print("logistic_map_sonify ready.")
#%% md
### Quick Test
We'll choose `r=3.8` or so to get some chaotic behavior.
#%%
logistic_map_sonify(r=3.8, x0=0.2, steps=20, alpha=12.0, base_freq=220.0, note_dur=0.2)
#%% md
## 3) Future Direction Snippet: AI or VR Concept
We won't fully implement VR or AI here, but let's show how you *might* generate an AI-based chord
progression for certain data clusters, as a minimal demonstration.
#%%
def fake_ai_chord_mapping(data_points, cluster_labels):
    """
    Pretend we have an ML model that clusters data into groups.
    For each cluster, we define a chord or scale.
    We'll just do something silly: cluster=0 -> C major chord, cluster=1-> F minor chord, etc.
    Then we play short chord stabs in sequence.
    """
    samplerate=44100
    note_dur = 0.4

    chord_map = {
        0: [0,4,7],  # C major triad (C-E-G in semitones from root)
        1: [0,3,7],  # C minor triad
        2: [0,5,9],  # maybe a sus chord or something
    }
    base_freq = 220.0
```

```
    # ensure cluster_labels match data_points in length.
    for i, val in enumerate(data_points):
        c = cluster_labels[i]
        chord_intervals = chord_map.get(c, [0,4,7])
        # let's pick a root offset from val in semitones
        root_offset = int(val*5)  # scale up data a bit
        wavesum = 0
        t = np.linspace(0, note_dur, int(samplerate*note_dur), endpoint=False)
        for interval in chord_intervals:
            semitone = root_offset + interval
            freq = base_freq*(2**(semitone/12))
            wavesum += 0.2*np.sin(2*np.pi*freq*t)
        sd.play(wavesum, samplerate=samplerate)
        sd.wait()

    print("Fake AI chord mapping done.")

print("fake_ai_chord_mapping ready.")
#%% md
### Minimal Demo
We'll craft some random data points from 0..5, a random set of cluster labels 0..2, and then hear
the chord stabs in sequence.
#%%
data_points = np.random.uniform(0,5,10)
cluster_labels = np.random.randint(0,3,size=10)
print("Data:", data_points)
print("Clusters:", cluster_labels)
fake_ai_chord_mapping(data_points, cluster_labels)
#%% md
In a real system, these clusters might come from e.g. PCA + K-means on a high-dimensional dataset,
and we'd pick chord sets accordingly, or generate them with an AI composition model. The final code
would be far more sophisticated, but this snippet shows how we can embed a *fake ML-driven mapping*
into our sonification pipeline.

## Conclusion
This Chapter 9 notebook underscores how *case studies* (brachistochrone, logistic map) can reveal
the power of Helical Sonification or multi-axis mapping, and how *future expansions* (AI chord
decisions, VR) might further revolutionize how we experience data.
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "43b869bc",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 9: Case Studies & Future Directions (Notebook)\n",
    "\n",
    "We'll:\n",
    "1. Demonstrate a **Brachistochrone** curve example, mapping its (x,y) param to pitch/timbre.\n",
    "2. Show how to do a small **fractal** or param.\n",
    "3. Outline a mock \"AI approach\" concept or a VR concept (we won't do full VR here, but talk about a function that might generate relevant data)."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "86bee00b",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
```

```
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "Chapter 9 Case Studies imports loaded.\n"
    ]
   }
  ],
  "source": [
   "import numpy as np\n",
   "import sounddevice as sd\n",
   "import ipywidgets as widgets\n",
   "import matplotlib.pyplot as plt\n",
   "print(\"Chapter 9 Case Studies imports loaded.\")"
  ]
 },
 {
  "cell_type": "markdown",
  "id": "3b1db1f4",
  "metadata": {
   "pycharm": {
    "name": "#%% md\n"
   }
  },
  "source": [
   "## 1) Brachistochrone Parametric Sonification\n",
```

```
    "Recall that the brachistochrone is typically parameterized by cycloid equations:\n",
    "\\[\n",
    "  x(\\theta) = r (\\theta - \\sin\\theta),\\quad y(\\theta) = r (1 - \\cos\\theta).\n",
    "\\]\n",
    "We'll pick a range for \\(\\theta\\) (e.g. 0..$2\\pi$). Then map $y(\\theta)$ to pitch, maybe $x(\\theta)$ to a secondary dimension (like timbre). We'll do a short code snippet to illustrate."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "04061813",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "brachistochrone_sonify function ready.\n"
     ]
    }
   ],
```

```
   "source": [
    "# Simple scale snapping\n",
    "def snap_to_scale(value, scale=[0,2,4,5,7,9,11], semis=12):\n",
    "    octave = int(value // semis)\n",
    "    rem = value - octave*semis\n",
    "    best = None\n",
    "    best_diff = 9999\n",
    "    for s in scale:\n",
    "        diff = abs(s - rem)\n",
    "        if diff<best_diff:\n",
    "            best_diff = diff\n",
    "            best = s\n",
    "    return octave*semis + best\n",
    "\n",
    "def semitone_to_freq(base, offset):\n",
    "    return base*(2.0**(offset/12.0))\n",
    "\n",
    "def brachistochrone_sonify(r=1.0, theta_max=2*np.pi, steps=20, alpha_y=10.0, alpha_x=2.0, base_freq=220.0, note_dur=0.3):\n",
    "    \"\"\"\n",
    "    We'll param from theta=0..theta_max.\n",
    "    Map y->pitch, x->some timbre offset (or brightness) for example.\n",
    "    We'll do a simple approach: we play a note at each step.\n",
    "    \"\"\"\n",
    "    thetas = np.linspace(0, theta_max, steps)\n",
    "    xvals = r*(thetas - np.sin(thetas))\n",
    "    yvals = r*(1.0 - np.cos(thetas))\n",
    "\n",
```

```
    "    # We'll map y to pitch.\n",
    "    # alpha_y scales y.\n",
    "    # x might scale amplitude of second harmonic.\n",
    "    samplerate=44100\n",
    "\n",
    "    for i in range(steps):\n",
    "        pitch_val = alpha_y*yvals[i]\n",
    "        snapped = snap_to_scale(pitch_val)\n",
    "        freq = semitone_to_freq(base_freq, snapped)\n",
    "\n",
    "        x_timbre = alpha_x*xvals[i]\n",
    "        # let's interpret x_timbre as amplitude of a second
harmonic.\n",
    "        # negative x won't matter much, we'll just do abs.\n",
    "        brightness = abs(x_timbre)*0.1\n",
    "        if brightness>1.0:\n",
    "            brightness=1.0\n",
    "\n",
    "        t = np.linspace(0, note_dur, int(samplerate*note_dur),
endpoint=False)\n",
    "        fundamental = 0.3*np.sin(2.0*np.pi*freq*t)\n",
    "        second_harm =
0.3*brightness*np.sin(2.0*np.pi*(2*freq)*t)\n",
    "        wave = fundamental+second_harm\n",
    "\n",
    "        sd.play(wave, samplerate=samplerate)\n",
    "        sd.wait()\n",
    "\n",
    "    print(\"Brachistochrone sonification done.\")\n",
    "\n",
```

```
   "    # Quick plot\n",
   "    plt.figure(figsize=(5,4))\n",
   "    plt.plot(xvals, yvals, 'bo-', label='Brachistochrone')\n",
   "    plt.title(\"Brachistochrone (r={:.2f})\".format(r))\n",
   "    plt.xlabel(\"x\")\n",
   "    plt.ylabel(\"y\")\n",
   "    plt.legend()\n",
   "    plt.show()\n",
   "\n",
   "print(\"brachistochrone_sonify function ready.\")"
  ]
 },
 {
  "cell_type": "markdown",
  "id": "bc1d9ccd",
  "metadata": {
   "pycharm": {
    "name": "#%% md\n"
   }
  },
  "source": [
   "### Quick Test with default values"
  ]
 },
 {
  "cell_type": "code",
  "execution_count": 3,
  "id": "9b0206b6",
  "metadata": {
```

"pycharm": {
"name": "#%%\n"
}
},
"outputs": [
{
"name": "stdout",
"output_type": "stream",
"text": [
"Brachistochrone sonification done.\n"
]
},
{
"data": {
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAdMAAAGHCAYAAADr4bhMAAAAOXRFWHRTb2Z0d2FyZQB
NYXRwbG90bGliIHZlcnNpb24zLjkuMiwgaHR0cHM6Ly9tYXRwbG90bGliLm9yZy8hTgP
ZAAAACXBIWXMAAA9hAAAPYQGoP6dpAABapElEQVR4nO3deVxUVf8H8M+wDaCggooiLu
4i7ihoZGG4vK45NojYmlFampmqW2amaSlqZX0WKaZv3ADl0oTMhANslRQU3LfhRQ1EFQ
QOL8/TjM6sggyw53l83697os7d87c+V4Y7nfuuWdRCSEEiIiI6LFZKR0AERGRqWMyJSI
iqiAmUyIiogpiMiUiUiIqogJlMiIqKYjIlIiKqICZTIiKiCmIyJSIiqiAmUyIiogpiMiU
iIqogJlNSzOrVq6FSqXSWWrVq4ckn8QPP/ygWFz169xdH//79H1kuPj4eKPUK8fHx5dr
/8uXLsXr16scLrhxUKmkhUmTZk8PepbPfu3YOPjw8+/PBDg77P6/893LZ1n/79Tr5jp2P
P9bc+ePAgevXqhapVq6J69o/YMmQIzpw5U2Tz6ZTz9+9FD4+Pl3Lt3T6f
MO++8g/bt26OwsPBxD4sMhMmUFLdq1SokbJGLW1NQYMGIDvv/9e6dBK1b5
9eyQlJaF9+/blel8JaF9+/blel1lJVNztXz5cty8eROvvPKQd9n//792LJlC1xcZ89yv/6vv/7
Ck08+iby8PGzYsAFff/01jQ0Z4+iby8PGzYsAFff/01Tpw4gYCAAFy7dk2n7AcsfSgAhxw/Jll8cjTsZYc4ncbvKC
j4sSJOuWmT5+Os2fP4ptvvvqnQsZEBCCKFrFq1SgAQf/zxh97r9+7dE7RKp/n68+RzU6dHB9Y/z/49d6f1u9zcc3mKTyGQJnp4eAMLUVHYJsAy9NRqNWqP6cfctP8+hTuYUd
z88Xdu3f1Hpe3t7fo16+f3ver0bJlSS9GjRw+WjQVVp3RNo2ZtQXegJg4ceJjvTYnJ0fP0ejHvXv3RN2
6dcXMmTMfWbaix1Bd37hxxowAg4uLiyvz6GiZo1a4ajgB7PLGzepcTdnPnzgtm1bW1vvxkYDhp6GxAQ+kbFsLmMcCSTW6ZkRDG0i1k7Ls6WOa
vaLdlZGQIe3t78eKLq8/+oPmBtCBDAFnWwpHzgJ7lfSAAA5ZSBUs4CGoFWXmeqTVFl3K2oNkdbgoxXaJSN3CiAmUKAC6YadW91e9iF
t7WFnZwdbW1vvttnPnzqzGIUmHhwoWYaNC1RrGVMb1gcnRcN7ELOOB2dn9xevvYzWhm6T
4uMDf3x9bt24/tsu/CwkJ8+umnnaNeuHRwdnsttHEzlIfWvVHV7ilXz6tvwIFz22ZtK1L2p59+Qvv27A
fHx98/fXXOs8/V8XV81w75w5LJiYSaeS6n4U7iGzNR3UvhDAAVZoFBzKnYdjw9g8fh
+/fra11+4cAGjR49vbx5ubyeBbKpxVk8i8NtPz1VFrSs2l5s2bw97aeR6tQeH6IZNEu72i
iYmKR4/j222/RvDOo6om3btkWq0qWM3Dhp5
c9bR8jwRcnUZ8/Hvo1oVzqMa3qrMp0w3e2fvxhDcfeVuvHGrnVhOZd6V8k2UKxNFJlL5XBHhAR
7P80m+aPpiVRATQjNdP5ML8fVdiDpvaWFr2D5/Cxc5N8ELoMR8pcfg8fwU1Ld1ZbZbu0fhbsn97mxLg5LU5+D4hzaVHDuNm3NbJ8u1tcJaTJBaJYoKCn4zzgtxw8XdU6JXPmgH11+KftcalxjwXWSbp4kbJK4+4M0ACDLjN5/9dc/B2kmIwpTFwZ5AVd35yoAGCDXyuz2EN4M7bXCl+PKB
iYmKR4/j222/RvDOo6om3btkWq0qWM3Dhp5cyWu9WDKlF1JnIGVqVWBMZj6HKbJc8OLuqhV9QQsIrwMI0mKl0wOq9imzXgDa/CF9TDtEhf8jnPSoUcEa3Bm9LxiyirPpLnfU6zkyFGqUiLT3g5ZO31eYDMfeIxi7hjIeAFs8nwnsFR+LvW7
0CzxUbIt9uAB1+h8/sonPvoTXZvUD9wAiDPt4u4PzEIqR/TATlT3LTIVhlIcdtd/Lx0gz6b9sVuM8GctmRH64sLr8bhSDwhhEnlbJ8WE92n5vt3xEmxgK0kN4JVYoDPSO1+hr2qQ06/Id9Lkhpji0vM+ODhAIF04ClWHmsIHVvOHyeE1FjT2lmvKFmDjL8uHkXhvahVhdoFn4bJEe2wxrWRuEN0yGKeXTL6OWBqjgdaVu2z9HUzqLb5Fpt+pfPnGjrnHm+HwcWjh64L82r2zP/+QDT+a+bwvIEVCwaYrTaQzKlrNHcJ7sHQTIl+7RuaBW/W25PDJy+1w/KNaUbtEV8KwKcGbpyfhm82M9WL/RQBDZzHcA+vLi5ZiFapd7MsYkQvwbKbkUH0GdbtNUHQHMnNVrUiHe95ReXi8jN/61Qq92S2v07j5vLHKkwm42SJ7Hw40s4nLe5bFuQWddTxEyhBDPZrJUCljXZ+oOSu4qTEGsLxLZWXgx6gl4bTeuLTYN1atd/6aO3MbHAqCE8hJYRQeTVMC9e9PQaawBMr
t7WnzQtCqSgWx3kpGSFBHoXrVMw1kjtYlEw7wOuwiH6gRY57Fu86vXN8gnPHAZCjU2Ldb4GkpkOWR/RhtVqb6uH0K7/Sz3hA7OXP1jcYd0dEzcxRMiJIbgMH3Bii9KJ68ZN7JZ8omJvBy39SVT5zBsMaGfUiVqWFANwUv9MFvXQVwLjBKLJFYwhsMeDDqdApgBNQYw+KyTpRqdKHRU9IYvCJkQsdLhGtMGX4pzXfmdhcI2s9VOBgXMCr4bMnUUyP6gX2YqbmhDHz24E02dB5RXH3bH4jM0NnsaZS38MfmzFFQmKsdOakwLqEUYhSqmLMgP+dE4OQr7cN4k6tC/C3DfiUoY9BbB0vMhbetu2Oa7P0b7Ydpr/MIgogHwcRVlBWvWQ3KzWv70qXqX90nGeUw7dMkTZSA1S2cGz1knMPrj8Ka/I93SZfQPT26zmrrMdD1+s1qs1jsuP1X7Mp+d9evrQ3b9Y6E+d3yJf3P8AtAvC+O+YYoOqxhy04YGGjnfJbDBOBMz9MH9S8TKc4TqjoL+T0kxz1NmsfS3Kxb2n3+ZEp6/hfQX7f17WZEXQyk5ExHj4G5RY5QG5HHyFjjlN6pIR4JBAbuazjuTrRpXOHyTbDOo8nrNi5X8S0dB0vt6F9Ov
iYmKR4/j222/RvDOo6om3btkWq0qWM3Dhp5
iYmKR4/j222/RvHlzoWsDvbLmNHk3Y7GbWtt20TIdO0UJ+FoW6NFBj7I9cCLVpAMhDZOuNsnvMwzOMfGqGAdAMmBYAJGQ6KrpB6SYK4tGvCVNVz/UEYA0Ahv4k05vNw0uFBDKbA5ICPbodHOCOkdtKq8dmEpg6zUyq7ebPhbuRhaCbQ0VIGKzo33lyUbmJYbrxQpECkAa5JYHDX0Aoob+WlSD/j8JUVyxzOAG9JrwUbzD7ZZKPpNsE9ZHIRSBQlU6Owk8OJ6JPWnNWLC3Dn8iQ6bFL5UbyyUbmfW/brv3DjBpv/2Ak+oo8WoTKJzGbV2nqdptdzDCoD
FMaw5PqSxqZXk39VsJlm1SkptmHDK+6aS0Hv79E9EM6kmI3ynOC3AXfFiizpkG46jYJ1Lfsf+sDo3lEc/77Cs5C9z8o2MjQg3yjSHXT6uk9m2pZx4HYtw6Yv1XPn0jfD4rQ5nwqQGDtl/6KYtu7i+LQ8L9RWfNNBzPLXV6+/dtxw66d6AGlEfTOqnq4v/1dWBbS0/u73jYGm4xLF+XrBzOb6vzUe3WdFD9cPbLVnqmvSMwZ6/9SlbkfbX3h8dDJ6r1xWIxTDfbNBKw+BTkn8dkJhuHY7I91Qgfa0X8gdh5jV3ZsY2SWu0GfnaDS9Eo+iLVzQ7LH6EanlvgYv1OJlpEe8m1uA8kMu0Pw67Vm+CtWX3l1Hpt8pyaP91hsoumh+Eijy7Vy3bBYIrR3GWCwtnU56XZk+1d/IVn7BUhhPX+gWv8VtWgfR8dRwPyq6omDvkELW2jSG9ULfhkZNjeyVzO0AZTFlzJlY+NKxZBnP2F8Vj3XhtKk2OtXDjZvEc+GeH2lr3YoDRyqcJbEwqn2/EePPGcuNl9dhJlwMKlBrH4uu09b99IuDBbbjxnpfx1fTstyQnLbG3X3dHdgBtGAs6ZLZp7eTH5HqRT1QdPBB4/t1MPv1QoV9CVl4nCRtJfzXGplhv1A/+66YXJ5jwfwu1zgqRSP8uM+WcD7BNN3ZCl6hoUjsQnM
iYmKR4/j222/RvDOo6om3btkWq0qWM3Dhp5
iYmKR4/j222/RvHlztW7GlIb27t0lAkXyOJrEL1XNufhdjpIYd9i3SbxnZcu6ThCKdNnVv/7nnHIoYmR7C+lkD49Y6ebrumXK8m3tPS4Ci4Nl1wS0MiRjpF07p6JtfS8RGlwXt+yjGjiVf98rphCLqHrBPcpqNyA9ZXeHQ8gpFV4eTFxkzvV66tG0W54TT2DFvHmdGWdy2edmBHK8ySqwJLL
PBNmQZFs+UJb5kXRS1hewmQz3Nrl3U72lLBW4uHVOPsaFqw3qDMEBlsZoowDUSQpS/X6ojGAmMaz1B2Y8hKL0XFfdbJqhehBE9cTqS/8eRo1bBhv59PAIZLkWlAKgtfxiyO9b8crYLk1dlA5cTj0YEnfcg+CYIOmX7W8wtatlpySTXj7iIKNH03W3YgR6ArfzQ3lPBCI3b6W5RzzN/ZZ9gDsSKVYYzVOObLTjx91otoZv14VxlKf48B+r/vwmzx4X1AFjQhVCxFixD0b6QyCMzXL7OYrF5hsvFEM7qcvmxNhJ8PUWi4YRmeTK00jT6AaA4tLp5MAjdtPtJpSRe8RMC/FD0j7XKG0yeDfhhCH6dXkeoZGoDfXjZxhtCamP8E0I9Z0C8nQnrUzhS7FL0ijMXq6eXWrSX5rYjAi/3ReBQlZU26JF0/UCm9LmD0rJjzWeVDCT5n8sNUcKIxYA+rkBEDxrJYZJWDCf6Q5Z7dz2IiMd5yEg2tV0yH4bnnaTAcyl6UmOUrQfvsEzEPnj0T7FtHyGTVcvMTk9aWSvJObh3WSqSqiu+5e5FVHNkRCbyAbS1s1XcfZyUFH2sg+O0HTKyKFIuhCsrtxB/AZ3KT8bRjP+ggUwhYmF8IxLLIEBBMjXNC5Cnyq4UfobEhUzpC1eaQBFlw13CXiCZMVv8qGHUgZDkYybncqz9YJ+T+PwEvuv7o+tz34brDg3uECIKM2djpTqbCcpkcUBcvAlNDGrfhwbQu9ywfDlUvS2c8vrpa+7Dc0hhDEBsj3G/B/aZk95jaCUPg8hsvk1fZ8e18t1bEm70QtQCrrTgELGJr3gchvhS3zZS
iYmKR4/j222/RvHlzoWsDvbLmNHk3Y7GbWtt20TIdO0UJ
ZAAAACXBIWXMAAA9hAAAPYQGoP6dpAABapElEQVR4nO3deVxUVf8H8M+wDaCggooiLu
4i7ihoZGG4vK45NojYmlFampmqW2amaSlqZX0WKaZv3ADl0oTMhANslRQU3LfhRQ1EFQ
QOL8/TjM6sggyw53l83697os7d87c+V4Y7nfuuWdRCSEEiIiI6LFZKR0AERGRqWMyJSI
iqiAmUyIiogpiMiUiUiIqogJlMiIqKYjIlIiKqICZTIiKiCmIyJSIiqiAmUyIiogpiMiU
iIqogJlMiIqKIDCZTIiKiCmIyJSIiqiAmUyIiogpiMiUiIqogJlMiIqKYjIlIiKqICZTIiKiCmIyJSIiqiAmUyIiogpiMiU
iIqogJlNSzOrVq6FSqXSWWrVq4ckn8QPP/ygWFz169xdH//79H1kuPj4eKPUK8fHx5dr
/8uXLsXr16scLrhxUKmkhUmTZk8PepbPfu3YOPjw8+/PBDg77P6/893LZ1n/79Tr5jp2P
/8uXLsXr16scLrhxUKmkhUmTZk8PepbPfu3YOPjw8+/PBDg77P6/893LZ1n/79Tr5jp2P
P9bc+ePAgevXqhapVq6J69o/YMmQIzpw5U2Tz6ZTz9+9FD4+Pl3Lt3T6f
P9bc+ePAgevXqhapVq6J69o/YMmQIzpw5U2Tz6ZTz9+9FD4+Pl3Lt3T6f"

NmoUGDRrAzs4OdevWxcSJE/HPP//o7ENTNf6o3xkApKen46WXXoKnpyfs7Oy0VZr5+fl
Fyj5s27ZtuHz5MkJCQsp8DI/LyurxT4/5+fn44Ycf8Mwzz8DJyUm73dvbG4GBgdi8ebN
2208//YS7d+/iueee09nHc889ByEEtmzZorM9JCQEJ06cQFxc3GPHR/pno3QARAUFBcj
Pz4cQAn///Tc++ugj5OTk4Nlnny1SdtmyZWjatCk+/vhjODk5oUmTJsjNzcWNGzcwffp
01K1bF3l5efj5558xZMgQrFq1CmPGjNG+fuzYsVi7di3GjRuHuXPnws7ODgcPHsS5c+d
03ufQoUN47bXXXMHPmTLi5ueGrr77CuHHj0LhxY3Tv3r3EY+nbty8KCgwcOFC1KtXDxk
ZGUhMTNQmn82bN2Po0KFwdnbG8uXLAQBqtRoAcO3aNXTt2hV5eXl4//33Ub9+ffzwww+
YPn06Tp8+rS2fn5+P4OBg7NmzB1OnTsVTTz2F/Px8/Pbbb7hw4QK6du0/EH3/
8gblz56Jq1apYuHAhBg8ejOPHj6Nhw4Y6sQ8ZMgQjR45EWFgYcnJyIITAoEGDsGvXLsy
aNQsBAQE4fPgwZs+ejjaSkJCQlJWljL+vvLD09HZ06dYKVlRXeffddNGrUCElJSZg3bx7
OnTuHVatWlfpZ+fHHH1G7dm20aNGi2OcfPgYAEEKgoKCg1P1q2Njo55R4+vRp3LlzB23
atCnyXJs2bRAbG4u7d+/+/C3t4ef/75JwCgdevWOuU8PDxQs2ZN7fMafn5+qFq1Kn788Uc
89dRTeomX9EDZC2OyZJpq3ocXtVotli9frFrfP7wv7b4NmzAoBo1KiRyMvLK3W/+fn54t69e
cuHHC19dXuz0hIUEAEG+99Vapr/f29vbi/Pnz2m137twRrQ4tNjnT4ULtHcGnvsDOCTp10to8dOxYRERH45zdf0LRpU+zYsQP
29vZ4/vnnH7nPPwMBAVKtWTfvYzc0NnnmmmSLxaN7/QcODc0NtWvXLvY4nnnnmmSLxaN7/QcOGDcPzzz+PXbt24YU
XXtBuL8vv7IcffkBYCDq1KmjjU60bHByM6BM6dOnY/fu3Y3LAlStX0LF/gYAGD
AgAH4448/SnyNIalUqjI9I9Mh6NTkjT5sdq5cmOq9LwqV8bsmKh4TKakuObNm6NDhw
vYPTo0ahevbr2OQ8pjKvj46OxvDhCh3hw46bPyvj46OxvDhwzFs2DC8/vrrcHd3h42NDWrVqtVRw3mq1Aag7Ty+AmS3W1tZ
wd3d/ZEyurq5FtqnVaty5
kteJcv35d5/6pRp06dbTPa46jTp06ZUrS5TmOh3+4DWrVqqWzWzXaVSwd3dXRt
Ped7r77//xvfff1/sFyJAfpkqzZ07d3S+ID3qGAD5JcfZ2bnU/eqqb5nfx8o8O8IAG7cuAG
VSqX9XLu6uuLu3bu4ffs2HB0di9Mrsg97e/tSP4tU+ZhMZhitfKbTj6Mh/2z1XtYS
u1Ir7lr527Vo0aNAA69ev13k+NzdXp1ytWrW...

fxNy5c4utEi2Jq6trsVXQj3L79m1s374dAPDbb78BAHbv3o2MjAxUqVJF++UMABo3bgw
AOHXqlHbbe++9h44dO6J///6YOXMm7t69i3fffRc1a9bEa6+9pi3n4uKCt99+G++88w5
cXFwQFBSEP/74A3PmzMH48eOLJM3r16/j5MmTBh+0gspJ6RZQZLmKa2Xp7Ows2rVrJxY
vXqwzIIOmNe9HH31U7L4+/PBDUb9+faFWq0Xz5s3Fl19+qW3h+aCCggLxySefiFatWgk
7Ozvh7Ows/P39xffff68tU9KgDT169NBphftwa96///5bjB07Vvj4+IgqVaqIqlWrijZ
t2ohPPvlE5Ofna1937tw5ERQUJKpVqyYACG9vb+1z58+fF88++6xwdXUVtra2olmzZuK
jjz7SGUBACNlS9t133xVNmjQRdnZ2wtXVVTz11FMiMTFRWwYlDNrg7e0tQkNDtY81v6d
r164VKXvnzh0xY8YM4e3tLWxtbYWHh4d4+eWXxc2bN4vssyy/MyGEuHbtmpg8ebJo0KC
BsLW1FS4uLsLPz0+89dZbIjs7u8g+HnTq1CmhUqnEhg0bdLaXdgyPS/OZK2558G8mhD
+h7cJIcT+/ftFz549haOjo3BychKDBg0Sp06dKvb9li5dKpo2bSrs7OxEvXr1xOzZs4t
tub5y5Upha2sr0tPT9XGYpCcqIfR0k4CIqBIMGDAA+fn52tbZliYgIAD16tXD//3f/yk
dCj2AyZSITMqff/4JX19fJCYmPlYjKFOWkJCAoKAgHDt2rMigG6QsNkAiIpPSqlUrrFq
1Cunp6UqHUumuX7+ONWvWMJEaIV6ZEhERVRCvTImIiCqIyZSIiKiC2M+0GIWFhby5Qq
qVatW6piZRERk3oQQuHXr1iOH8GQyLcaVK1fg5eWldBhERGQkLl68WOqAIkymdxAMSH7
x4kWdiX2JiMiyZGVlwcvL65ETVTCZFkNTevkMRkSkREj7zlxwZIREREFaRoMg0PD0f
Hjh1RrVo11K5dG4MGDcLx48cf+brdu3fDz88P9vb2aNiwYbFTW0VFRaFFixZQq9Vo0aI
FNm/ebIhDICIiUjaZ7t69GxMnTsRvv/2G2NhY5OfnIygoCDk5OSW+5uzZs+jbty8CAgK
QnJyMN998E5MnT0ZUVJS2TFJSekaMGIGQkBAcOnQIISEhGD58OPbt21cZh0VERBbGqEZ
AunbtGmrXro3du3e/fuxZaZMWMGtm3bhtTUVO22sLAwHDp0CElJSQCAESNGICsrS2c
g7D59+qBGjRqIjIx8ZBxZWVlwdnZGZmYm75kSEVmwsuYDo7pnmpmZmZCUDO71eSpKQkBAU
F6Wzr3bs39u/fj3v37pVaJjExsdh95ubmIisrS2chIiIqK6NJpkIITJs2DU888QRatWp
VYrn09HS4ubnpbHNzc0N+fj4yMjJKLVPSwNjh4eFwdnbWLuxjSkRE5WE0XWMmTZqEw4c
PY+/evY8s+3ATZU1N9YPbiytTUtmPbNmYdq0adrHmn5FREopKAD27AHS0gAPDyAgALC
2VjqqymPpx0+mxyiS6SuvvIJt27YhISGh1BEmAMDd3b3IFebVq1dhY2MDV1fXUss8fLW
qoVaroVarK3AERPoTHQ1MmQJcunR/m6cnsHQpMGSIcnFVFks/fjJNilbzCIkRdIkREd
H45dffkGDBg0e+Rp/f3/ExsbqbIuJiUGHDh1ga2tbapmuXbvqL3giA4iOBoYO1U0kAHD
5stweHa1MXJXF0o+fTJeiyV6YTJ07Ed999h61bt6Jat6Wraq0lnZ2c4ODgAkFWwly9fxpo
1awDIlrufffYZk2bhhdeeeAFJSUlYuXKlTivdKVOmoHv37vjwlSYAEGDhyIrVu34eeffy5
TFTKRUgoK5BVZce3rNdefFGWM8cqz4IC4OWXSz5+lQqYOhUYONB8jwHZDkqqVThUYONA8j59MnFAQgGKXVat
WacuEhoaKHj166Lwu6Cjs7O1G/fn0RERFRZN8bE7a2tsLHx0dERUW
VOa7MzEwBQGRmZj7uoRGVW1ycEDJllL7ExSkdqWHw+MkYlTUfGE0/U2PCfqakhMhIYNSo
R5f77jtg5EjDx1PZLP34yTiZZD9TIkvm4aHfcqbG0o+fTBuvTIvBK1NSQkEB4OMjG9sU
97+lSiVbtZ49a573DC39+Mk48cqUyMRYW8vuH4BMHA/SPF6yxHwTiaUfP5k2JlMiIzJk
CPDjj0Ddurrb69aV2829n2VJx+/paRnHT6aL1bzFYDUvKa2gAPj2W+D554Fq1YDr1wFra6Wjqjw7dwK9egGDBwNvvAF06cIr
UlIGq3mJTJi1NdC7t1zPzgbM+fbFlGRv/0t/XHgrLp77bzyczeHS+5UnA1yYqT6S/bPrEKP6MNMNKODfbFdiGDalyKnvd1IpvcA+7B/kAu2u
FAhdWnHJaXT+0PUDR3yJpwaEVlQ8T5pyCEI2wLrKAfw6ia3I4yALSbKKbLeAUZFcFmtwn2chQZfwd4Np/aovkEaB0A1Aiz2jaA+kpWCJY41cbjBqhyzxY+wIaLT1ERPMMrkqPiYUWy8xyEAhVhsB/pgoMnFxicV2qFd5PVaYHUkzrrNbOT7lLRTkD+zJsfNoZbQqWFE6kvZp/Y8H3pzfRKCSSAZR2Vx8Qk4+dfnDxDXvrv7Y/IOMW2MbUvP33HBDhVkNpe3S1CcKHQ24pY0LOAbDgVdDf0Mz5TGh3ZQuBoTWvdCRIeEzAfEw5xyHBlQSlx0KL6IKtfRbn43IQb7v44jvQfiyHvoQIGX/lU2Jc/Xp2qHZAA3NO+jEhLiXV8U6BxLzBKD4fVIhzxLT7ZHm26k6lVImtsrhmtrifCTV1mF6mBrjnjyJIcwVAnCDVbXVTRW0NsDiqW9kAlBx4sLhDqBMLxvHPXN8Ssu/rc5LIhsHF3BaSwSf+DmNFrndcSfhB3XLZQeGjfWGmYN3gHqvyjWVXrWdSOAbSVEOSq9HDbrH8/Dpqqp6tXKUnsmRBP0tiwjmC5UvWzYgNWJA12lOH8WTl7P4OrnB/pqVMTlD1L4Wcvs1zL0KZNbTsqoyaoWgdqYuRuZWvLaSJvXOoxWJMmW4MNCQr1LM59yCSk/wuDT/9u5lmCq4ChcrmWOjBUpG2oZwqXD1yDWKAJD/iDL8xpbu7szqaZ2ew1hHyWEEpR8H7GdqotT9JGMGi6uPQ2hYUZlQWP97/vgShKE1axsXXw5QlA6Y3tI7n13Wj2PA+Z4AyZr9/gL9NQSA1uGczbhANCsj3wVEA7kn6bFAvvEFCaVdKCYnrEARVNmQB3kctGSjpnM+HYpR5YQZ43gGG3S4hFLL6TJ20fbmPTxwHmAgmZkAd0RLQCPQ8FdFk98AdYwiy5C6rHlASZ5ll6qJk9HRFcCcFK2lGlFNkJhFGx2F3/2h4yCZL39hQxvQzyU0LeFLyHh5L67EbQAGgDhw5ZZjEd5zdMPU6B+Vg8JlCAGvy8j+3AHZzc6N3eF5KrhG/1TS+HeIw4w/Y/8WORr8yxx28ir9+AeLfPuWYnFTZMVj7/6vHrcfGBrYkk9xz17ZQ4NqIE6JmKwp9dyY0tMY3w9YdFDsDxOvU17+ykTrgF/HJsp5PLzuycfSd8bFRhh7k0Il3pBpOKRT8XSa0C8Vug9f8DUhB4tJ7Ptmg1ywXbuU0W7Wp2UEL6Gz5yzoKQdOR0HBPNOKq0WNmwS+DvyHcPyBC0sOyAtJa7xM9uMRDk+08Fqo3DRYBvZGhd3umBF3BUVvqlwHqsuLDp8P4wqUdnLU1nOD1qKWG60NCPqcVyP1yAeDcJqWyldW3X6SYGmrO5NQy/Z8tK/4kZfugDzX8ZHvFUcEXTMJo/ZZpQBz7zeKMRq4vo3MUdnNvPn5EG8LajE6d2c4g3ZhFCCpt2vF5U/

NHw7k58v7qgMGABkZSkdG5ozJlEhPfv9dVi1GRQG2tsAnnwBbtwL/zllPlcjZWbbw/d/
/ZF/U7duBdu1kDQGRITCZElWQEMDixUC3bnLc2AYNgF9/lXNvciAB5ahUcv7XffuAZs3
kBOOBgcC8eXJQCCJ9YjIlqoDr14H//Ad47TVZpTh0qKzW7dhR6chIo00bYP9+YMwYee/
0nXdkq+r0dKUjI3PCZEr0mPbulVWHP/wAqNXA8uXAhg2yipGMS9Wqcjzf1asBR0fZz7d
dO+Dnn5WOjMwFkylRORUWAuHhsq/opUtysu7ffgNefpnVusYuNFRepbZqBfz9NxAUJK9
U8/OVjoxMHZMpUTn8/TcQHAy8+aa87/bf/8qTc7t2SkdGZdW8uWws9uKL8n73vHlAz57
ynirR42IyJSqjX36RSTMmBnBwAL7+Ws72Uq2a0pFReTk4yJa+kZHy75eQALRtK1v9Ej0
OJlOiRygoAGbPBnr1ko1WWrYE/vhDTlLNal3TNNkk7JPavr1sTNavH/DGG8C9e0pHRqa
GyZSoFFeuyCQ6d66sEhw3TlYRtmypdGSkL40by1GTXnlFPv7oI6B7d9nNiaismEyJSvD
TT7LqLz5etgZduxb46GpTMi1otx/WNjgaqV5cNynx95Ri/RGXBZEr0kHv3gJkzZUO
jjAx5n/TAAdnYiMzb4MGyn3DnznJoyCFD5Cw0ublKR0bGjsmU6AEXLgA9egALFsjHEyY
ASUmy+wtZhvr1gT175Ji+gJwftWtX4NQpRcMiI8dkSvSvbdvkVWhSEuDkBGzcKGd/sbd
XOjKqbLa2wMKFwI8/yrGVNY2U1q9XOjIyVoom04SEBAwYMABl6tSBSqXCli1bSi0/dux
YqFSqIkvBL6ehzX6tgyd+/eNfDRkKnKywNefRUYOFBOKN2xo6zqGzpU6ChIaX37
wnXJOTk465dLS0mDPywsqxunTcoD6JUvk42nT5DCBDRsqGhYeP99eT999GjgjyBGGQwc5HnNoqNLRkTEw6Xu
mK1euRK9eveDt7a2zPTs7G97G97e3vD09ET//v2RnJxc5DD5untXNiwaPlw
m0m7dZFUeEyk9Sq9e8rPSsrP0Ssydw+zYwdqxcsrMVDowUZ7LJNC0tDTt27MD48
16tXYtm0bIiMjYW9vj27duuHChQuys5GO3HjsDhw0pHRkpSCSGE0kEAsmp28+bNGD
HY9GiRby5Qrs7OxxKLFdYWIj5Gjj27duje/zuje/fuaaMiU5ysLDx
O5ToOMl5r1wJhYUBODlCrlhxXt3dvpaMiU5aQAIwaJUfKsrcHli4FXniBw0ay6ysLDg
7Oz8yH5jklakQAl9//TVCQkJKTaQAYGVlhY4d5Z6ZapWq+Hk5KSzkPnIyQGefx4
rgYHAoUNMpFRx3bvLat/Lat/gYHn74KWXgGeflbcPyLKYZDLdvXs3Tp06hXHjxxyrBACKSk
p8PDwqITIyNgcPQp06gSsWiWr5Nl57N57D4iNBr5h0A/qwYNKR0a0C0a
VSdFkmp2djZSUlKSkpAAzp49i5SUFFy4c4AEAMGvWLIwZM6bIRZ5
77733sHPnTpw5cwYpKSkYN24tBjIeMiBLBypbyXdeyYTJ67dgHvvgtYWys
dHZkbKytg+nQ5cpK3t+xy5e8vR08yjhtpZGiKJtP9+/fD19fDT19+cX9ZvR6B19cX777
7LgDZyEiTWDMzRFRVV4lXpP//8gxdffBHNmzdHUB0NG49Y
t2X1h/HjZub53b1kV9+STSkdG5q5fq1mLFzngx6BBcjP7L43byodGRMiZLveFFFF
Mxic5WXZ5OXVKXoF+8IEcY9XKJG9okKkSAvjsM3m1mpcnr1bXrZPJlkyLWTdAInqYEHI
c3S5dZCL18pItLWfMYCKlyqdSyflRExOBRo2A8+flkIQffwUFiodHRkCTzNkUgoK5Py
ikZHyZ0GBnCpr6FBg0iR5FfCf/8hq3a5dlY2VyM9PNkQaMQLIz5e1JAMGyKn9gOI/z2S
aFB1OkKg8oqOBKVOAS5fub6tdW16VXrt2f6aPKVPYz4+Mh5OTTJY9e8p7qNu3y0nnX34
Z+N//dD/Pnp6yr+qQIcrFS4+H90yLwXumxic6Wl59lvRprV1bdk/o2LFy4yIqjyNH5D3
9v/4q/nnNl8BNm5hQjQXvmZLZKCiQV5ulfe2ztZV9+4iMWvG+/AY6OxT+v+YxPnco
qX1PDZEpGb88e3aqw4ly+LMsRGbvkZDlIfkmEAC5e5OfZ1DCZktFLS9NvOSIl8fNsnph
MyeiVdeg/DhFIpoCfZ/PEZEpGLyAAqFOn5OdVKtmvNCCg8mIielwBAbLVbkktzvl5Nk1
MpmT0rK2B5s2Lf05zQlqyhGPukmmwtpbdX4D4EoQ/DybIiZTMnp798pB6gE5Q8eDPD3
ZjYBMz5Ah8nNbt27xz5fU2peMF/uZFoP9TI3HvXuAr6+cSm3cONNnJfc8e2TjDw0NWhfE
bPJmqggLdz/OWLfKqtWFD4M8/AQcHpSOksuYDJtNiMJkaj4UL5fi6rq7A8ePyJ5GunUeuV

L3tK4fBl4+23g/feVjog4aAOZvPPn5WTegBwgnImUzF21asCyZXJ9wYKSR0oi48NkSkZ
r8mTZub17dyA0VOloiCrH4MFAv37yFseECZxc3FQwmZJR2roV2LYNsLEBIiI4cD1ZDpU
K+PRTeb80Lg74v/9TOiIqCyZTMjrZ2XIuSEBOWdWihbLxEFW2Bg2Ad96R69OmATdvKhs
PPRqTKRmd996TY5PWry8bYRBZotdek42Rrl0DZs1SOhp6FCZTMipHjgCffCLXP/uM/e3
IctnZAV98IddXrJCzzZDxYjIlo1FYCISFyb53Q4bIRhhElqx7d2DsWNkIKSwMyM9XOiI
qCZMpGY2vvwYSE4GqVe8Pt0Zk6RYuBFxcgEOHZMMkMk5MpmQUrl0D3nhDrs+dK4cJJCI
5hOaCBXL9nXdkewIyPkymZBRef122WGzb9n5LXiKSnn8e6NoVyMkBpk5VOhoqDpMpKW7
3buCbb2T/ui++kH1Lieg+Kyv5v2FtDURHAz/+qHRE9DBFk2lCQgIGDBiAOnXqQKVSYcu
WLaWWj4+Ph0qlKrL89dCYW1FRUWjRogXUajVatGiBzZs3G/AoqCLy8oCXX5brL74IdIm
ibDxExqp1a9nnFAAmTZKjg5HxUDSZ5uTkoG3btvvjss8/K9brjx48jLS1NuzRp0kT7XFJ
SEkaMGIGQkBAcOnQIISEhGD58OPbt26fv8EkPFi0CUlPlfaHwaHwcKWjITJus2cD9eoB584
B8+YpHQ09yGhmjVGpVNi8eTMGDRpUYRYn4+HgEBgbi5s2bbqF69erFlRowYgaysLOzYsUsO
7rU+fPqhRowYiIyPLFAtnjakcZ84ALVsCd+8C334LjB6tEREexm/rVmDQIHk7/bWVlZZOgsZlhC
R4Zj1rDG+vr7w8PBAz549bDdRpUYpn4+HgEBgbi5s2bqF69erFlRowYgaysLOzYsUlhC
yquruXSAwEPjvf5WOiMg0DBwI/Oc/ss/pyy9zIHxjYVLJ1MPDAytXrkRUVBSio6Pf
z9OzZEwkJCdoy6enpcHNz03mdm5sb0tPTS9xveHg4ngk6Ghgxw7A1hZ
YvpwD2ROVx7JlcnSwPXtk4z1Snkkl02bNmmHcuHHo0KEDAgICsHTnz3YnNmuGFF15A+/bvcI9ddfB65
pHjozCyGKbHvQrFmzkJmZqV06dmaWfolLBeQHsDc+bI9ddfB65
fVzQcgokl0+J06dIFJ0+e1D52d3cvchV1XSL8lerD1Kr1XByctJZyHBmzwYuXwYaNgT
efFPpaIhM09SpQKtWQEYGMHO0mtGQys6fT5ORkeHh4aB/7AaB/7+/sjNjZWp0xMTAaMxMTAy6du1a2aF
RMZKT7w8V+Pnncs5GIio/W9v7A+F/9RXw66/KxmPpFO0en52djOSvDV5JhlBpBjnDdhYXA8OFAnz5KR0Rk2rp1A8aNa1a
ulP9bBw/KJEsKEAqKi4sTAIosoaGhQgQQQkNDRY8ePXRes3jxYtG4cWNRpWrpF1/10M1wNa
hLl9WOhoi85CRIYSrq/zfWrhQ6WjMT1nzgdH0Mzum7Geqf3//DTRbRmymreyZOVjoj
IfKxeDTz3nGzhe+yYbKBE+mHW/UzJ9EyfLhNp+/bAhAlKR0NkXkJD5dyt2/fbynt2/fbylPlYv
JlAzul1+AtWs5kD2RoahUr4KnWVqUjsjxMpmRQubn3B7KfMAHo0sjM2HEcZ6IjMVcu
WsgYIkNMYZmcrG4+lYTIlg1q4EDhxAnBz48DcRIb2zjtA/fpyAvG5c5WOxrIwmZLBnDo
FfPCBXP/kE6CEuQmISE8cHQHNJFyLFwNHjigbjbjVhMiWDEAKYOFFW8/bqBYwcqXRERJa
hXz9gyBDZr/vll2W/JbjlMyiI0bgZgYQK3mQPZElW3JEqBKFtkq0qpVSkdjWvSkdjGZhMSe8
yM+W4oQAwaxbwwNztRFQJvLzu3zL2jZl47EETKakq1ZmKeqhfq+8A8n
MtG0L3LghEyoZFpMp6dWBA3M3/c4P3e+3M79WBA3IAeBWYrtkXaSpKaWysZH9ulUqSP9M+kwEwmZLePDi
Q/ahRwNNPKx0RkWX0gV48UW5/sILwK1bysZj7Zhnan4Sms0vWVXzfCJSXng4UUKu
WHLOX/5eGw2RKepGWdn+i7/nzAXd3ZeMhIqlGDWDRIrk+dy5w9qyy8ZgrTgnTgKw
sOVzgSy8pHQ0RPWj0aODJJ4E7d4BJk/QcJ95gfYTdYN06wYYYN06wMpKVvVaW4WFQRZE9SKU
CIiLkxOHbtwObNysdjeVlA2jQdAwnDq0bIHgf8kF6c2RlS35JTxT/s9ThtsvX7qaFjq0FmKlDPSQGnPmJVaZ
ZUoV8+KEcg9fDA3j/faWjIaLvSvPkmQ0AhcPkyGTmTeS6fY5K2YZYFwZTewTnSiggCwdClQyiT
0RGQEHBzu9wNfuhRISvE0HPZFVOQtPCOBHmBBOqD/NIREVFVKQd0IMBI8Okoad0OBm2y+cKo7
JlB5LZCSwa5cc4eizzzzZiQPZEp+eQToFo1YN8+4MsvlY7GPDCZmsX98w/w6qty/ds3h7nQ
NFA2HiMqpbl1g3jy5PnMm8PffysZjGDjpmMMqdzeegu4ehcnVXoA/XWloyGixiGixzFhAuDrK78
c8/+44phMqVx+/132VWW/l32VwPkT7VazXiI6PEOBD+t98CcXFKR2R2YEy2PZfzKLzZYMFIYCQECA
wUOmIiKgiOnWSA+ABOAD8mdurrLxmDIdz9HkpOB6tWBjz9zoOoi6o0cGDx6CPPNIPgDc3IDx4x4G

PPlI6GtOlaDJNSEjAgAEDUKdOHahUKmzZsqXU8tHR0Xj66adRq1YtODk5wd/fHzt37tQ
ps3r1aqhUqiLL3bt3DXgk5u/yZeDtt+X6hx8CtWsrGw8R6Uf16rJ1LyAbJZ0+rWg4Jkv
RZJqTk4O2bdvis88+K1P5hIQEPP3009i+fTsOHDiAwMBADBgwAMnJyTrlnJyckJaWprP
Yc5bqCnn1VSA7W86P+MILSkdDRPo0ciTQs6es5uVA+I/HRsk3Dw4ORnBwcJnLL1myROf
x/PnzsXXrVnz//ffw9fXVblepVHDnHGB6s2MHsHGjHMD+iy/kgPZEZD5UKmD5cqB1a+C
nn4BNm+TADlR2Jn1aLCwsxK1bt+Di4qKzPTs7G97e3vD09ET//v2LXLk+LDc3+LDc3F1lZWTo
LSXfu3B/IfsoUoG1bZeMhIsNo2hSYNUuuT5kip1SksjPpZLpo0SLk5ORg+PDh2m0+Pj5
YvXo1tm3bhsjISNjb26Nbt244efJkifsJDw+Hs7OzdvHy8qqqM8E3CBx/IyYQ9PTkwNpG
5mzkTaNwYSEsD3nlH6WhMi0oI46gdV6lU2Lx5MwYNGlSm8pGRkRg/fjy2bt2KXr16lVi
usLAQ7du3R/fu3bFs2bJiy+Tm5iL3gTbhWVlZ8PLyQmZmJpwsePT21FR5JXrvHhAVBQw
ZonRERGRosbFAUJC8nfP774Cfn9IRKSsrKwvOzs6Pzs6PzAcmeWW6fv16jBs3Dhs2bCg1kQK
AlZUVOnbsWOqVqthpOTk85i6TQD2d+7B/TrBwwerHRERFQZnn5aNkgqLJT9ygsKlI7
INJhcMo2MjMTYsWPx3XffoV+/fo8sL4RASkoKPDw8KiE687F2LRAfL6ds+vRTDmRPZEk
WL5ZTKu7fD/zvf0pHYxoUTabZ2dlISUlByr+T6p09exYKSm4cOECAGDWrFkYM2aMtnx
kZCTGjBmDRYsWoUuXLkhPT0d6ejoyMzO1Zd577z3z77z3s3LkTZ86cQUpKCsaNG4eUlBSEhYV
V6rGZshs3gNdek+vvvvgs0aKBsPERUuTw8gPnz5fqsWUB6urLxmAShoLi4OAGgyBIaGiq
EECI0NFT06NFDW75Hjx6lLhdCCiKlTp4p69VdmZqYAIDI
zM/VwlKbnxReFAIRo0UKI3FyloyEiJeTnC9GhgzwXjBqldDTKKWs+MJoGSMakrDeDecz
iItCtm1zfvRvo3l3ZeIhIOQcOyPF7XSmHUDJDKM/Pz7g16PHctESmTp/Pz
u9zOfMAHgqKwlYzIlrWXLgMOHARcXYOFCpaMhImPw/vvyHuqpU8Y7yBCCBUpHY7yYTAkAcPG
ibGwEyERaq5ay8RCRcXB2BjQdQv9oJidZlaqIIC2fUlMlL+nDwZyMmR90ufe07
p6IjImAwbBvTuDereere/Hzd8wf7oio80D0pIzpajr156ZLudisrICKCA9kTkS6VSs5
n3LIl8PPPPcv7TGzfuP+/pCSxdatmjpPG0aWGio4GhQ4smUkC22GMVDhEVp1Gj+yOhOPZh
IATnf8dCh8vxiqZhMLUhBgbwiLakzlEoFTJ3KKhsiKqqgANizp/jnNOcUSz5/MJlakD1
7ir8i1RBCNkQq6R+GiCzXnj3yCrQkln7+YDK1IGlp+i1HRJaD54/SMZlakLKO9c85AYj
oYTx/lI7J1IIEBMhWdyXNAKNSAV5eshwR0YN4/igdk6kFsbaWzdeLo/kHWHDEHWbJEliMietC
D54+HEyrPH0ymFmfIEGDNmqLbPT2BTZssu58YEZVuuBB5nqhbV3e7Sg7e7hwfPH0ymFqiwUP7
08gL+7/+AuDjg7FnL/kcgorIZMgQ4d06eN0Z3ZRzHc+cyfMk6kFWrdO/hw3Dnj2WeDJJy2
3aoaIys/aWp43Jk2SjzdsUDQ4co8BkamEyMoDYWLk+cqSysRs+XN4+XN4v3btX9jG1ZEy
mFiYqSg5S7esLNGumdDREZMo8Pe+33l2/XtlYlMZkamE0Vby8Vby8KiUifdcCcSzTnFkvFZGp
BLl8Gdu+W6yNGKBsbsLEZmHoUPdQDByx7ogwmUwuycaMoF9Oo
l1y356pTJ1IJERsqfrOIlIn3SnFMiI0uelcrcMZlaiDNngN9/lxN/DxumdREZE4GDwb
s7IDUVODIEaWjUQaTqYXQtLDEATc3ZWMhItMi7Mz0LevXLfUql4mUwuhqeIdNUrZIj
IPGnOLevWWWZVL5OpBTh6VFa92NrK6hgiIn3r3x+oUkUOTfr770pHU/kUTaYJCQkYMGA
A6tSpA5VKhS1btjzyNnbtfnx374afnx/s7e3RsGFDREREFCmzadMm+NnZwc7ODn5+fti8
eXMlHYFpaGQTuwDw1OgIDB8p1TU2YJVE0mebk5KBt27Z4//33y1T+7Nmz6Nu3L55++mkk
JycjbVBwpoIVENpz3wC2mOuyZdjL2zw6v2xRZwN2bIGxTHzn4fJ/XBEdArQDKN8+7gC68BmYiTbNb2I6MGHr
3LwICApCcnIw333yTkt6bxKE9r/7EiD6c2vPZOl1Rw0bhX72Nmz6Nu
C3E+mrOIlIkPStOrdsAEoKFA2lsqmEsI4ardVKhU2b96MQYMGlVhmxmxzowZ2LZtG1JT7X
bwsLCcOjQISlJQEARowYgaysLOzYsUNbpnfv3nqnBmJsRCpaI5nwCZIfQb3mLFzPH8Cb6t0d+DTjOLanbaT2IMdnS6P9PjYpWkKSsOSmQKMsEPnJR
jHZCR2L8f6NgRcHAArl4FqlZVOiIiMld5eeyCbG/DPP8Avv8gGjaurPnApO6ZJiUlISg
oSGdb7969syxt7dy7x8e8x8TJRFnWVZZZWc884xct7R
WveVOpmPHjkVCQoIhYnmk3NyNmk9PR0uRuLm56Xn4n7Dw8PhOysXby
8vPQfvAIKC+93ieFADURUGTTnmk2bgH+vcSxCuZPprVu3EBQUBQUHmCZAkRNnNmDDDm+/Pm4fPmYIeI

qkUql0nmsqaV+cHtxZR7e9qBZs2YhMzNTu1w0k7mEfv0VuHQJcHICgoOVjoaILEFgoKz
qvXHj/nSPlqDcyTQqKgqXL1/GpEmTsHHjRtSvXx/BwcHYtGmTtqrVUNzd3YtcYV69ehU
2NjZwdXUttczDV6sPUqvVcHJy0lnMgaaaZfBgwN5e2ViIyDJYW98fZc2Sqnof656pq6s
rpkyZguTkZPz+++9o3LgxQkJCUKdOHbz66qs4aaCpA/z9/RH70FedmJgYdOjQAba2tqW
W6dq1q0FiMlb5+XJge4BVvERUuTQ9BzZvBu7cUTaWylKhBkhpaWmIiYlBTEwMrK2t0bd
vXxw9ehQtWrTAJ5988sjUlBRcuHABgkx15eUlBRcuHABgKx+HTNmjLZ8WFgYzp8
/j2nTpiE1NRVff/01Vq5cienTp2vLTJkyBTExMViwYAH++usvvLFiwAD///DOmTp1akUM
1Ob/8Aly7BtSsCfTsqXQ0RGRJunQB6tUDsrOB7duVjqaSiHLKy8sTsmTmZtEv369RO2trb
Cz89PREREiKysLG2ZyMhIUb169Uf4sRuKy4uTgAosoSGhgohhAgNDRU9evTQeU18fLzU
VdnZ2on79+iIiIqLIfjdu3CiaNWsmbG1thY+Pj4iKiirXMWZmZgoAIjMzs1yvMybPPSc
EIERYmNKREJEleuMNeQ4aOlTpSCqmrPmg3P1Ma9asicLCQgwdOhQJCQlo06ZNsPVjPSg3
30b59e5w9e7bCyV4Jpt7PNDdXNgDIzJSTgXfvrnRERGRpkpOB9u1le42//5YNIU1RWfO
BTXl3/Mknn2DYsGGwL6VFS40aNNUw2kZqDn36SibRuXeCJJ5SOhogsUbt2QNOmwIkTwLZ
twOjRSkdkWOW+ZxZFXmZZEsYaxenmrNTE4O8P33cp2
teIlISZpzUEwMcP26srEYGpOpmdm2Dbh9G2jUCOjQQeloiMiS+fjI6t78fOCB+UjMEpO
pmdF0kh45UlazEBEpSXN1au4DODCZmpGbNwHNZDms4iUiYzBihPwZHw+kpSkaikExmZq
RzZvlwNKtWsmFiEhp9esD/foCL7pX/iZ8lqBwZKlMzlefwO7dsl
1TbUKEZExGD5cdtP77TfAXIcgYDI1E5s2yflLO3YEGjdWOhoiovvc3YEGjdWOhoiovvc3YEnn5TrmjmWzQ2
TqZnQdIpmFS8RGSNzH8CBydQMXLggJwJXqVjFS0TGacgcqgQwNYWOHwYOHZM6Wj0j8nUDGh
ayAUEyPF4iYiMjYsL0Lu3XDfHql4mUzOgqTbRVKMQERkjzW2oyEjZVcacMJmauBMngIM
HAWtr4JlnlI6GiKhkAwcCDg7A7AyZNyijZzwmRq4jT9tnr1AmrVUjYWIqLSVK0K9O8v182
tIRKTqQkTglW8RGRaNFW969fL7nzmzmgsnUhB0+DPz1F6A4MGKR0VRVKsGVGj9e0LVKsGXLw
IJCYqHY3+MJmaME0Vb9++rVEb+gLOzsrEQEZWFvT0waJBcN6e+dCZTE1VUBGzYINfZipeI
yrlNzwGRqovbtA86dA6pUnuX9Dn4jIFPTsCbi6AlevAnFxSkejH0ymJurbb4HCQlnF4+urdDRERPx
sRETlYWsLDBsm182ll1qpfJ1AQVFNwfQYRVvERkijTnrqgoIDDDX2Vj0gcnUUBCUkAOnPX
q94fnIiIyJU88AdSpA2RmAjt3Kh1NxTGZmiBN39JnngHs7JSNhYjocVhb35Z+YwxwGcFA
8mS5fvhwNGjSAvb09/Pz8sGfPnhLLjh07FiqVqsjSsmVLbZm1a9cWWyY3N9fQh2I0mlUKjvTZ1
nq0UADtRARKZNU9W7bRuQk6NsLBWlaDJdv349pk6dirfeegvJycno2bMnh3aWKpvC5k/jc/HAwa
5pUuXIi0tTbtcvHgRLi4uGKa5k/0vJycnnXJpaWmwt7ev8jJ9/Bm2cA3Nc+pJ9k9v+J0A22S0R
kijp2Bo2BG7fBn74QeloKkbRZLp48WKMGzcu4PzOzs2bhp8mxehGyAJPg
wd3fXLvv378fNmzx3HPP6ZRTqQV65dzd3SvcFjp7+HBTZTUJZKpUKt2ZZEyZYysk
0Ly8PBbW4cQFBQkM72oKAJJZxjKmVK1eiV69e8Pf3P93b21on////kh+xPQ
Eubm5yMrK0lmM0Z07wYJYtcp2teInIHGhu3+HJVGvy=3YAfzzj6KhVIhiyTQjIwMF93PT2e7
m5ob09PRHvj4tLQ07duzA+PHjdbb7+Phg9erV2LZtGyIjI2Fvb49u3brh5MmTJe4rPDw
czs7O2sXLy+vxDsrAfvwRyM4G6tUDUnRROhoioopr1Qpo2VK2B9m8WeloHp/iDZBUKpX
OYyFEkW3FWb16NWrUqIFBgwbplOvcuTNGjhyJ1q1bo2fPntiwYQOaNm2KlwkjOsuBRvF7
6Ir45BD43w3qVLF4wePRPpt27ZFQEIzAAkuizL2qj2Y8BgDUnRROhoioopr1Qpo2VK2
CZmamdrl48eJjHYuhPTh8oJXifzkiIv3QXJ2a8gdzCsnJDRMsHHl1ytfo
wIQS+/vprhISEwOBO4RfUOsrKzQsWPHUq9sLEkOBpTo9nLcMTPA7RkWUo0BbZEVSkXnRdJHZtUs
OMWiKFEumdnZ28PPzQ0xMjM72mJgYdO3atdTX7tq1C6dOncK4ceMe+T5CCBw+fBgeHh6PFa8x
C8Spt61bg7l2gWTOgXXuDGeTm4Cpdm1OnbBEPHX8M4Mt4CTZqnF
OLVrP
1NRWvvvvoqLly4gLCwy+nXMmDFFXrdYFXrdyFKmo4tuzVq1KvLc
BuHHjkJKSot2nqXwircMteBERCbF1Kt6bZR88xErVD69euYO3u0tLS0KvK2zfvl3
bOjctLa1In9PMzEiHvzkxeERUVh6dKlxe7zn3/+wf79+4+6v7777k0qlBPsjWYzvvoj09HQ4OzvD19cX1cXXcQXeOa9gR93M2Nz6cX9quvu7dp+wzP5GCDLg+N4hqJ9XPx6PEQh9NO1IKR0hCvTCRtpMWyYXsvwUYoMy8eTqnrMu8pHTwwpr
bOjctLa1In9PMzExERUVh6dKlxe7zn3/+wYsvvoj09HQ4OzvD19cX1cXXCQkJ6NSpk8GPx1C

uXwdiYuQ6q3iJyBwNHw5Mnw7s2SMnDjfSdqAlUgkhhNJBGJusrCw4OzsjMzPTKO6frlg
BvPSSrN59RC8fIiKT1aOHHHv844+B115TOhqprPmAbUJNgKYzM69KicicmfIADkymRu7
KFWD3brmuafFGRGSOhg6VI7sdOACU0gHDKDGZGrmNGwEhAH9/oH59paMhIjKcWrWAXr3
kumbOZlPBZGrkNNUdnCGGiCzBg1W9ptSih8nUiJ09C+zbJ0c7emhiHCIiszR4sJyn+dg
x4M8/lY6m7JhMjZimv9WTTwJmNPENEVGJnJ2Bvn3luik1RGIyNWKaZMoqXiKyJA8O4GA
qVb1Mpkbq2DHg8GHA1hYYMkTpaIiIKk///kCVKvJW1++/Kx1N2TCZGinNVWnv3oCLi7K
xEBFVJkdHYOBAuW4qwwsymRohIXTH4iUisjSac9/69XIAfGPHZGqEDh6UHZYdHO5/OyM
isiRBQUD16kBamhyv19gxmRqRggIgPh6YPVs+7tcPqFpV0ZCIiBShVgPPPCPXFy+WLXv
j4433KpXJ1EhER8sRjgID708CHhcntxMRWaK6deXP778Hnn1Wnh/r1zfO8yKTqRGIjpZ
jUl66pLv9xg253Rg/OEREhhQdDbz/ftHtly8b53mRU7AVozKnYCsokN+0Hk6kGioV4Ok
pm4hbWxs0FCIio2BM50VOwWYi9uwp+QMDyJa9Fy+axg14IiJ9MMXzIpOpwtLS9FuOiMj
UmeJ5kclUYR4e+i1HRGTqTPG8yGSqsIAAWfevUhX/vEoFeHnJckRElsAUz4tMpgqztga
WLi3+Oc0HackSNj4iIsvx4Hnx4YRqrOdFJlMjMGQIsHGjnLf0QZ6ewKZNHOieiCzPkCH
y/Kfpa6pRt65xnhfZNaYYldk1RuPSJVltYWUFfP014O0tqzCM6ZsXEVFlKygAdu+Wc5z
m5gJHjgCtWlXe+7NrjIlJSZE/W7QAQkPlhOBMpERk6aytgaeeAvz85OMjR5SNpyRMpkZ
Ck0zbtlU0DCIio6Q5N2rOlcZG8WS6fPlyNGjQAPb29vDz88OeUnrhxsfHQ6VSFVn++us
vnXJRUVFo0aIF1Go1WrRogc2bNxv6MCrs0CH5s107RcMgIjJKmnOj5lxpbBRNpuvXr8f
UqVPx1ltvTk5GQEBAQgODsaFCxdKfd3x48eRlpamXZo0aaJ9LikpCSNGjMCQkIfPy5Gh
EkJAQDB8+HPv27TP04VSI5tsWkykRUVGac6OxXpkq2gCpc+fOaN++++fOaN++PSIiIrTbmjdvjkG
DBiE8PLxI+fj4eAQGBuLmzZuoXr16sfscMWIE8Du22Pn36oEaNGoiMjCxTXJX
dAOnWLUDzNlevArVqGfwtiYhMyu3bQLVqQGGhHPnI3b1y3tfoGyDl5eXhwIEDCAoK0tk
eFBSExMTEUl/r6+sLDw8P9OzdE3FxcTrPJSUlFdln7969y9xnYrdKKydJbKpLmhXqc
OEykRUXEcHYGmTeW6MVb1KpZMMzIyUFBQADc3N53tbm5uSE9PL/Y1Hh4eWLFiBaKiorBp
0yY0a9YMPXv2REJCgrZMenp6ufYJALm5ucjMzNRZKsvXX8ttSpcCTOwqGCWvSf5BAQUISb4
dHY1mzZqhZ8+eSEhI0JZT08v1z4BIDw8HM7OztrFyZv8v1z4BIDw8HM7OztrFy8urAkdWfqziJSJ6NGOu6rVROgD
VQ8NbCCGKbNNo1qwZmjVrpn3s7++++Pixcv4uOPP0b37t0fa58AMGvWLEybNk37OCsrq1ITaTKlvyEhE9Wtu2wLp1xplMFbsyrVmzJqytrYtXVpunTpT2ofu7u7u
farUaTk5OOktlYkteIqJHM+YrU8WSqZ2dHfz8/BAbG6uzPTY2Fv7+/mXeT3JyMjw8PLSPu3btWmSfO3bsKHWfarVaZu5t3yk
B/v7+RfYZEx+xNTrn1Wpvx84PBhuc5kSkRUMs058sQJ2SDJmChazTtt2jSEhISgQ4cO8Pf
3x4oVK3DhwgWMGzcOgO1/9/WvP+WeA/Q8A5ZCvO9G00o+sn5tnOVJIn0TnwQYaVlmh6Xkl6Px1OeYXGjZ/FiQpPG8RZ7TZvqr9z
ftUAeTHdC9e4dqV3SiI8NZCMO4HktP0/gLJ0d9dEvVHTw7TQeSY3Xze/RvTeRDeahXX6jmx9sNdGmpVFIVg5ORdVw1HOvSBKNzh
AzQ34+2/gzz+BTp2Ujug+RZPpiBEjcP36P36dcydOxdpaWloOXlo1aoVtm/fDm9vvbOWQCRb3E8wBAWlqaTp/
TvLw8TJ8+HZcvX4aDgwWNatmyJJH3/8EX379tWW6dq1K9atW4Aaht4e3334b77zzdHd01aoT169qe
jc+fOlX58ZaGp4m3dmsMHEhE9Stu2QEyMrOo1pmTKge6LNbYCA06eJ2G2xP
RUTFmzAAWLgQmTAA+/9zw72ijSxd0Ctk6zQ6IncQEM7NNS0NDV87ewTJ4xr19XlDmDwy9otj3Dj5g8yGHNdDn2ErKuxLYcOydG
QjAWTqYLS0+WiUsl7pkREVLqmTQF7eyAnBzrMyyVZDmqrRpU9mal4iiSmdjc//
iw5iGFWQyVRBHPiIiKj9jnNuUyVRBHPmIiKj8jLEREpOpgtiSl4io/IxxonAmU4XcuQM
cPy7A7XmUyJiMquTRv589MgIrT8R58uIeeCFENF8uefsll3rVqVNN8ktEZE5qFbt/vvdsEaE5qFbt/vCrxnJ1ymS
qkAereEuZHY6IiIphbFW9KTKK4f1SIqLHZ2yNkjJhMFcKRj4iIHp+xdY9hMlVAYSFw+D8w7ZkD
RVYTm3JmaCuTmKhoKACZTRZw5A2RnA2o10KyZ0tEQEZkeT0/AxQXIzweOHVM6GiZTRWi
uSlu1kkNjERFR+ahUxlXVy2SqADY+IiIyHGNqhMRkqgA2PiIiMhxjaoTEZKqAi0cMhnUiyKiijOm7jFMpgqrg
Hr0yFUDISJtNKd/26HAILuD8kFhERlZ+PD2BrC2RmAufPKRlk2kl01RHNGwIODkWs
RkSmzswNatpTrSt83ZTKtZKziJSLSH2O5b8pkqrrPJcK1nsh8N0dlFSdiMI0JDJkUTZMM+ZCmZb
uMUymlSg39/5IHUymREQVp7kyjPXCOO+Ocf5eJgMq1Eqaly6KvqPEvL6WjISIyfTVqAPX

qyfXDh5WLQ/Fkunz5cjRo0AD29vbw8/PDnj17SiwbHR2Np59+GrVq1YKTkxP8/f2xc+d
OnTKrV6+GSqUqsty9e9fQh/JInMOUiEj/jKGqV9Fkun79ekydOhVvvfUWkpOTERAQgOD
gYFy4cKHY8gkJCXj66aexfft2HDhwAIGBgRgwYACSk5N1yjk5OSEtLU1nsbe3r4xDKhX
vlxIR6Z8xtOhVdJj1xYsXY9y4cRg/fjwAYMmSJdi5cyciIiQHh5epPySJUt0Hs+fPx9
bt27F999/D19fX+12lUoFd3d3g8b+ODjtGhGR/ln0lWleXh4OHDiAoKAgne1BQUFITEw
s0z4KCwtx69Ytrg/Lzc1FVlaWzqJvQtz/Q7NbDBG
R/mjOqX/+Cdy7p0wMiiXTjIwMFBQUwM3NTWe7m5sb0tPTy7SPRYsWIScnB8OHD9du8/H
xwerVq7Ft2zZERkbC3t4e3bp1w8mTJ0vcT3h4Oj
ERBarfn05olxeHvDXX8rEoHgDJNVDLXGEEEW2FScyMhJz5szzB+vXrUbt2be32Ll26YPT
o0Wjbti0CAgKwYcMGNG3aFJ9++mmJ+5o1axYyYMzO1y8WLFx//gEqgqeJt0UIOgUVERPp
hZXX/6lSp+6aKJdOaNWvC2tq6yFXmz9vUYN24cNmzYgF69epVa1srKCh0
7diz1ylStVsPpJyUln0TdW8RIRGY7SIyEplkzt7Ozg5+eH2NhYne2xsbHo2rVria+
E2LFj8d1336Ffv36PfB8hBFJSUUuDh4VHhmCuCLXmJiAxH6UZIirbmnTZtGkJCQtChQwf
4+/tjxYoVuHDhAsLCwgdDI6tfLLy9jzZo1AGQiHTNmDJYuXXpor2odnz
E2LFj8d1336Ffv36PfB8hBFJSUUuDh4VHhmCuCLXmJiAxH6UZIirbmnTZtGkJCQtChQwf
4+/tjxYoVuHDhAsLCwgDI6tfLLy9jzZo1AGQiHTNmDJYuXYouXbpor2odHBzg7OwMAHj
vvffQpUsXNGnSBFlZWVi2bBlSUlLw+eefK3OQ/2IyJSIynAe7xwhR+X35FU2mI0aMwPX
r1zF37lF37lykpkpaWhVatmwJ59O7y9vQEAaWlpOnOONm3b
qAMA///yDF198O8Eenp6yB2doavry8SEhLQqVOnSj1jVvLxcL4ZYSO8eSD5eY6VPrCC
RAVy5AtStW7nvrxJC6fnJjU9WVhacnZ2RmZmpmpl/une/YA3bvLIQRLGI+CiIgqqFUr4Oh
R4IcfgDLcBSyTsuYDxVvzWgGRZA2soAD46Se5Xr26Fe5fExERPrXpo38+f3
3QHx85Z5vmUwNKDpadibevl0+/vZb+Tg6WsmoiIjMT3Q08NFHcn7ymBhI
dDQwdCly6pLv98mW5nQmViEg/NOfzbjAzd7ZV5vmUDpGJUtAFSQYH8RvRwItVQqQBPT9n
C19q6YrESEVkyQ59BJQXv2lPyHBWQfqIsXZTkiInp8xnK+ZTI1gLQ0/ZYjIqLiGcv
5lsnUAMo6cqHCIxwSEZk8EzIPDnFpMqAQOBAQULlxWCWq0COAMkeWhHlAS+nBGoC
1NbB0afHPaf7gS5aw8RERUUULW9eL59OKFW5WvlZdmQ7ZYmUQ59j8zZuDjW
EsdAsSkFBAe7du6uK2tYWNjWunx6CboNkaqzPMtu8YUQ59j8zZuDJw
+Dbz7ruxEHBDAK9LKkp2djjUuXLoEfcVKSo6MjPDw8gAi1cnjhBaBGDeD
atYqfb8uaD3hlamA3bsifI0IcN1rxIpzFGOm2G4slKSgowAwrg4JpgtBDWxyoAqnRACeXl5uHb
tGs6ePYmTZrAyop37kXFPAvm8nUgPLzgZGsZp3bqrq7KxWJp79wen5BCIF
atWrBwcFB6XDIqzQj4OMDW1O9Qt1lZbnHxl4e7O3tN0kGpEm
kAODiolwcloxpKQ0Xo1WLgcHwiN1FRRl+/Xnnmvy7+AWnBcMrLy50esTOfOnYNKpmUJKKbM
hGXSZVbxERJWnZk358+GB7w2JydvrUqYrGrV1aqi2Mr169ZWfHW/6Ntm6xRw+6S
qr169GtWrV6+0mIrz5JNPIjs7G2+//TYKCIQVbr2I2lkcqmmrkadBkerGpMikoo+VRWsYmFOtwBzAB
xUKlU2sXV1RV9H9cOfcGc4iAvVbDNPVT5GNh+gpVy1qtBXYnBrRMooDRTvxFxiWY
lpSEtLQQ27du2xpLqcfDxbBfdDhf4w4CAQM7C0/CbL7rhaZvmm9fcN2mkjeJ3P5g5
s2IAnn3WSM5sbwaE62Sawr7e1Wx+zAmWtzE28/96JHo1GZf9GoK0tPXSdG+8
o4J0yZoDATcgDxfRLeUhyCp2uzZyMp+5ZY6HYkTyZjQxYZQCFA1DV1UjXGLzYXrmMrNoVVYDZM
fvx/x8fF47rnKmKEmQZbNeeGqHGtGThG5LnhTlV9ThM2eJLGMdc2PDcuFJi3JN0IfxVnLkd0cbzoW8i
jBxwdHVGjRg307t0bbNx9oGl3YWIg33ngDLi4ucHd31Q6VS4KAB0oD66vczR9W2fvwDJjFh0t1UBVpc3bA2D
3DwAQERGBRo0awc7ODs2aaNDs2aNcO3335b5VdF8nnaHT07A1DH07ds
XVatWhZubG0JCQpBRmdsKhMlrkwhqIjMzEwBQGRmZoP+PGCQEIMW+engKjMrtz544
4duyYuHPnjhBCixs+bdQYsnOLnvcoaGhYuDAgdrHtdHrt27dEi+ihKCgoEGfPnhU
ARP369UVUVJQ4c+aMuHz5srh06ZL46KOPPRHJysjh9+rRYtza2ZL46KOPRHJysjh9+rR2Fr/99pt2X2Ya

oUaOGWL16tTh16pTYs2eP+PLLL4UQQrtfHx8f8cMPP4jjx4+LoUOHCm9vb3Hv3j0hhBC
rVq0Szs7O2v21bNlSjB49WqSmpooTJ06IDRs2iJSUFJGbmyuWLFkinJycRFpamkhLSxO
3bt0SQgjxn//8RzRv3lwkJCSIlJQU0bt3b9G4cWORl5cnhBAiOTlZqNVq8fLLL4uUlBT
x559/ik8//VRcu3ZNCCFEjx49hJOTk5gzZ444ceKE+Oabb4RKpRIxMTHauACI2rVri5U
rV4rTp0+Lc+fOiejoaGFrays+//xzcfz4cbFo0SJhbW0tfvnlF53XeXp6iu+++06cPHl
STJ48WVStWlVcv35dCCHElStXRM2aNcWsWbNEamqqOHjwoHj66adFYGBguT6LZHhLl8r
/veHDK76vsuYDJtNi6CuZDhwo/6AREfqJi8rOlJOptbW1qFKliqhSpYoAIDw8PMSBAwe
EEPeT3pIlSx65r759+4rXXntNCCFEVlaWUKvV2uT5MM1+v/rqK+22o0ePCgAiNTVVCFE
0mVarVk2sXr262P09XFYIIU6cOCEAiF9//VW7LSMjQzg4OIgNGzYIIYQYNWqU6NatW4n
H1KNHD/HEE0/obOvYsaOMtmHDhom+ffvqvO7tt9/tt9/
WPs7OzhYqlUrs2LFDCCCHEO++8I4KCgnT2cfHiRQFAHD9+vMSYmUwr39q18n/vqacqvq+
y5gNW8xoQq3mNh6MjkJ1dtmX79rLtc/v2su1P04G8rAIDA5GSkoKUlBTs27
Oxvnz57VlOnTooPOagoICfPDBB2jTpg1cXV1RtWpVxMTE4MKFCwCA1NRU5ObmomfPnqW
+d5s2bbTrHv9OAHn16tViy06bNg0j3jx49Hr1698OGHH+L06dOl7js1NRU2Njbo3Lmzdpu
rqyuaNWuG1NRUAEBKSkq5YtTE+XCMD/9+UlNT0a1bN51t3bp1075vcfuuUqUKnKpW5ZsqUKqlWrpt3
3gQMHEBcXh6pVq2oXHx8fAHjksm3FkzTMAe6Vo1IBVaq
UrWxQkJxt4vLl4u93qlTy+aAgw/xNq1SpgsaNG2sf+/Sn+/n5dnZGV9++SXGjx+vLfOgRYs
W4ZNPPsGSJVUvQunVrVKlSBVOnTkXev1MWlXVIRVtbW+26cVZ5CYsOmTMHzz77LH7
88Ufs2LEDs2fPxrp16zB48OiySbh4LIbTvVZY4H4xRE+fDMT78+9GUK+l9y7LvwsJ
CDBgwAAsWLCiybw9DzzxN5aK5tX/hguzSVhkTjCh+Zbp8+XI0aNAA9vb28PPzw549e
tv3v3bvj5+cHe3h4NGzbEF198UWRfmzZtQosWLWBnZ4cWLVrgyy+/rNyDICJ6sOoFrMVFQUWrRoAbVajRYtWmDz5s2GCr9Ymi4VFy/Kx9O
mGb5LBemPscyPeP89VbCyssKdO3dKLLNnzx4MHDgQo0ePRtu2bdGwYUOcPHlSu7xJhZVbbjUOdhj1NmjSBg4M
Ddu3apdfYmjZtildffRUxMTEYMmQ1Vq1aBQCws7NDIe1P8gYT8oWHArN5lK9xPhWwUOdclu0aIH8/Hzs27dHu27fDyPdqzkqAI+cjt+/
bj2k/6Jg7EpbLNZ3rHhW4aFAw2Vrzy5wh0sl9AbBxLQuF4sTN8N2gZeoeDCfgsPS2NmY1+0ioG1BCyB1bdGaWDBx8a0jnzPxtVZ3p+GZWOqmqMjRbqbsJ3j4c/zvRIQDw2IKahXKC1rnKn9zYmq6YFSW6lLu7Re1cp8DPqbKg/iTC9mqWTyk/3vRf2K7XPX0kVl5TMRjhw
FQEAAkpOT8eabb2Ly5MmIiorSlklKSsIESMQEhKCQ4cOISQkBMOHD9f5BzYkJbtUkP5
o5kesW1d3u6enw3G7I+RFzc3ORnp6O9PR0pKam4pVXXkF2djZ2djjYGDBhQ4msaN26M2NhYJCY
mIjU1FS+99BLSo9O1z9vb22PGjBI44403sBbNGpw+fRq//vorVq5cWe5jCAkJQUJEJ
8fDzOnz+PX3/9FX/88Yc2ODdWvXx/Z2dnYtWsXMjIycPBPv2bTRp0gQDBw7ECy+8gL179+L
QoUMYPXo06tati8cffxwhAMTJkzA4cOH8ddffffyEiIqLCLWZff/11rF69GlJ
88QVOnjyJxYsXIzo6GtOnTy/zPihZOZIgbN25g1KhR+P3333MmzBnExMTg+eefL/LFgZS
hOf8+8NEHUEnn34rfnn18nTp1EmFhYTrbfHx8Yx8xMyZM4st/8YbbwgfHx+dbS+99JLo0qWL9vHw4cNFnz59dMr07t1bPPnkk2Xep6Ws2bMCwsLT+1kI+4iLZ+5N0awIgAjRE
L9vHw4cNFnz59dMr07t1bjBw5ssxxPW4DpPx8ITw9S26IPEF
xQnz3nfxp6L9daGioAKBdqlatKjp27Cg2btwohBA6tbU66t9lvqWFbV
r1xZvv/22GDNmjE7L4IKCgEdqz4tbG0aZLO7zssLy4uroKAGjBhUOe1tRb169tR169sLE48uroKAGL27NlCCCFu3LghQkJCRJUqVcqwdb0Bw1N2bCIGVDGl
UlJubK0aOHCm8vLyEnZ2dBgyw4b9vprRmhLb2bFR8e7JjhYmo7V9wo3UVvmO8Ou7Jl1/7jbGNu1xQ0ddSfdFU+5yJ9hfbrG
zs3BwcBC9e/cWJ06c0DmW+Ph40SlTJ2FnZyfc3d3FxIkTtWXhAZdmgR2oNnxmP5FrPVzdmrcf+U8ph8
QhIaGah8DEJs3by7yu12+fLlo2LChsLW1FWlpaWq/tPa81yNSgiUT9+ukYc4lWkaC9Rlk8x1k
MHjxYVK9eXTg4OAgfHx8xdepUVhYWOT9NNgAqXIY6vxr9K15c3NzhbW1tYiOjtbZPnnn5nn
yZNG9e/diXxMa2x2G8Yjn1ZBn
mZmoXTQu98ibTuLiyte7899xEBsQTGBkLfhYrh6HOv0bfmjcjIwMFBQVwc3PT2e7m5qZ
TPfWg9PT0YsvnPSPgEgPDwczs7O2sXLy+txDglpafotR0REZaP0+VfxBkh
laWH3qPIPby/vPmfNmoXMzEztclHTcqicytqgjw3/iIj0S+nzr2JdY2rWrAlra+sV4x

Xr14tcmWp4e7uXmx5GxsbuP7bmbOkMiXtE5DDt6nV6sc5DB0BAWXrUhEQUOG3IiKiByh
9/lXsytTOzg5+fn6IjY3V2R4bG4uuXbsW+xp/f/8i5WNiYtChQwdt/7CSypS0T30yti4
VRESWQvHz7+Pd6tWPdevWCVtbW7Fy5Upx7NgxMXXqVFGlShVx7tw5IYQQM2fOFCEhIdr
yZ86cEY6OjuLVV18Vx44dEytXrhS2trbalo5CCPHrr78Ka2tr8eGHH4rU1FTx4YcfChs
bG50xSh+losMJRkUVbVXm5SW3U+XQNPq4ffu20qGQhbt9+zYbIFUifZ9/jb41r8bnn38
uvL29hZ2dnWjfvr3YvXu39rnQ0FDRo0cPnfLx8fHC19dX2NnZifr164uIYga+3bhxo2j
WrJmwtbUVPj4+Iqqcv0V9jM1b2V0qSFdeXp44duyY+Oeff5QOhSxcRkaGOHbsmMjnSaD
S6PP8W9Z8oBKivBNEmb+srCw4OzsjMzMTk5OSodDj0EIgQsXLuDevXuoU6cXuoU6cOrKwUb2t
HFkYIgdu3b+Pq1auoXr06hxw0UWXNBxybl8ySSqWCh4cHzp49zNAPFFlq1690tzd3ZU
OgwyMyZTMlp2dHZo0aaId7J2ostna2sKaLQ4tApMpmTUrKyvY29srHQYRmTneSCIiqo
gJlMiIqIKYjIlIiKqIN4zLYamt1BWVpbCkRARkZI0eeBRvUiZTItx7do1AHjs2wAHi
83Lp1C87OziU+z0EbilFYWIgrV66gWrVqpc428yhZWVvLCxYsXzXLwBx6faePxmTY
eX+UQQuDWrVuPHPyFV6bFsLKygenqenp9725+TkZJYfdg0en2nj8Zk2Hp/hlXZFqsEGSER
ERBXEZEpERFRBTKYGpFarMXv2bL1MPG6MeHymjcdn2nh8xoUNkIiIcqIV6ZEREQVxGR
KRERUQUymREREFcRkSkREVEFMpgayfPlyNGjQAPb29ujQoQP6b29vDz4f3lvEhBISMGDAANS
pUwcqlQpbtmxROi9S9Cg8PR8eOHVGtWjXUrl0bgwYNvvEhISMGDAANS
/7NixQ+mwDCI8PBwqlQpTp05VOhS9mTNnDlQ7qlc7i7u6udFh6dfnyZYwePRqurq5wdHR
Eu3btcODAAaXDKhWTqQGsX78X78eU6dOxk5GQEBAQEAQgODgYFy5cUDo0vcjJyUHbtm3
x2WefKR2KQezevRsTJ07EEb7/9htjYWOTn5yMoKAg5OTlKh6YXnp6e+PDDD7F//9//37s378
fTz31FAYOHIijrR48qHZpe/fHHH1ixYgXatGmjdCh617JlS1y5ckW6Kh6YXnp6e+PDDD7F//37s378
fTz31FAYOHIijrR48qHZpe/fHHH1ixYgXatGmjdCh617JlS6SlpWmXI0eOKB2B2S3ty8eRP
dunWDra0tduzYgWPHjmHRokWoXr26qGVTpDederUSYSfHels8/HxETNnzlQoIsMBIDZ
v3qx0GAZ19epVAUDs3r1b6VAMpkaNGuKrr75SOgy9uhmjSpImIjY0VPXr0EFOmTFE
6JL2ZPXu2aNu2rdJhGMyMGTPEE088oXQYRlNUk3rbUxXhjBoVcvuwDMwmSmC2Qg9xPN6V
FFZGZmQkAcHFxUTgS/SsoKMC6C6/fHzV5UeHdlnX39ayY0qEJMQgyRNdLRpAn6C3LBUE8
yFPgdnUHBewVNLO/KfaZgmRuXLcZIzBUOxJzyyUHSvl4Sp2+ejAmAFDIacSjfb+qXU5YL
sRmlEYjKo3lVXXZ/wfQ0HiyUtUHGWwNs9E6JBdX4g/aXrPk6xdzJpZb9xMlBXeGI57Bo+/
9AobjGVXaXw2J0fXBIbSnAXzMehVE6tVbBmNKJ3yojMHHQ+vP4qI8zr+rk8FnSQNF3bMUL
ZHixJybSUNP6lkTc+HTBwjmFHX+9u4OnHSlJs9dlv2Kz/KDZ7Aby1Wv7fQjdlPVyF3EaL
T5LTduWjdN0RkNc8ejfvZ3tFZddE/EvDTAuTPRsUsAQuDcFDqQSt3gUU6FXv6+jDTTA
gwuGYcIDsTMcEzBXaXSMXXL6dVW0/7IIHeF0IHaO3HvHtHPIT3rtSN1V6/iUU+gPb8nwN
PG0zUI8QDbgS4GEf5N2zn0Ot+f1wLE1qUNs8xMwcQ5aFV/TPFe5Ubty2UOJtsn/OW6FpM7
SxoVvXiEZL6UOaJ4YkUYwUkH1K6cJe/ZUcdXZIT5qD4UaqfK0eTH8G0K0dHwsm0B2KGVK
BCJ7iAb/7VVJ4q1n5x1Z9ykPbSpsnzuuqPV/yWSrO5d+pgW6rN+CJkeBmVrVm70I8Ze9qQUv
q8uORzq5GQdUsNq2o1q8UWH4f7jmmtcWw9Gaf8BM7Ee3j+ZB6PoTMOLdhnbxJ6Et3JW+D5
zJX2lZT6r8ZX3FWkyZkxhNSfdJU68c4RY0e1l6AFyvPGYJZVlSg9rcQUbtpoKpHBLTR5
8PtE7jwx1O5Lqa8dlkvgyy8gn56lwoQx7SxtxYkqeLDa7o9yvDsjpcaFpBa8L2bdk4XG
v9ZfOA/g0dlNjZuxnx2akzuFCWzdT9e9ZGrDNcLjLxdNK+ARtmtDlvBQ2sFGMywgojqhcMI
d2ZhZKjNcm1vNUBWjHpbSQ5OsPQvWn8z1TsixhvIy4w7Dqc+ZrvQGHVeXvtaTLWMxf5uB0
jYqLGNUxiK5upTsfUmi0TH/jpXxGKnSFJ1OtAKNpjwPkKTT1xI9pBD3zzbnPp69xeMHbD0
zR1CpcEDGbi6UPWCBgIHPqI+ftlMSPqHfE8b1idJhw6chiClOTPW7Gt+mUK2yb7ge7BGDC
k9DCgLEbO0wHLTg/FPw4dGlxTmq5U/R8H47sPPu7kWcrXq8Uac4hUqG+M0ZIkr2/6MT5Y
xrJ8QAEY4s8T5Vh6EAbnzE/6Bd2OT+hhb0jbP7I2EfHNjYhp4SrN8hgW2/0mb0EJ4l+ptKW
o86l9t7HAFwgeT1sVVpn/bt+yK8YbkKjbk0pwgmt2DTHlS6HQN9CvGtIOHkPOxNaGh8Fr/7I
vIqgRhcj2zgL1ahfWfGTTHnNEtb+J01UOn/nOJM2Nf7tmFg3VbHdZfE5+bbPyOoKK3Z1x2Wg
tRZKvPoaeabd4JtIyP0yPM8NO9MqebIdKvsXrPDLnTnmQ+VhTzr53NZKbMhDWfu+gUd4Y
qUZkbbn2k0Jx3ZkD6fg5Vf+Kq4OS+b3H1pkMffS9P0C/7gHNekZRJuGmflONc8OX5cK/xl

7dmJmANi3bx/s7OwQExOjYGRExo0D3RORju3bt2PQoEFITEyEj48PfH190a9fPyxZskT
p0IiMFpMpERUxceJE/Pzzz+jYsSMOHTqEP/74A/b29kqHRWS0mEyJqIg7d+6gVatWuHj
xIvbv3482bdooHRKRUeM9UyIq4syZM7hy5QoKCwx/vx5pcMhMnq8MiUiHXl5eejUqRP
atWsHHx8fLF68GEeOHIGbm5vSoREZLSZTItLx+uuvY9OmTTh06BCqVq2KwMBAVKtWDT/
88IPSoREZLVbzEpFWfHw8lixZgm+//RZOTk6wsrLCt99+i7179yIIkLp8Ii7179yIiIkLp8IiMFq9MiYi
IKohXpkRERBXEZEpERFRBTKZEREQVxGRKRERUQUymREREFcRkSkREVEFMpkRERBXEZEp
ERFRBTKZEREQVxGRKRERUQUymREREFfT/mFlzwtd+TqgAAAAASUVORK5CYII=",
      "text/plain": [
       "<Figure size 500x400 with 1 Axes>"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "brachistochrone_sonify(r=1.0, theta_max=2*np.pi, steps=12, alpha_y=10.0, alpha_x=2.0, base_freq=220.0, note_dur=0.3)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "3107f96e",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## 2) Fractal / Parametric Example\n",

```
    "We'll do a small param (like a Lissajous or a logistic map) just to show how you might map 2D data to pitch & timbre again. We'll keep it short."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
   "id": "ba7ccb1a",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "logistic_map_sonify ready.\n"
     ]
    }
   ],
   "source": [
    "def logistic_map_sonify(r=3.5, x0=0.2, steps=30, alpha=12.0, base_freq=220.0, note_dur=0.2):\n",
    "    \"\"\"\n",
    "    The logistic map: x_{n+1} = r*x_n*(1 - x_n), 0<x<1.\n",
    "    We'll track x and sonify.\n",
    "    \"\"\"\n",
```

```
    "    samplerate=44100\n",
    "    x = x0\n",
    "    out = []\n",
    "    for i in range(steps):\n",
    "        out.append(x)\n",
    "        x = r*x*(1-x)\n",
    "\n",
    "    # Each value in [0..1], we can scale it up.\n",
    "    for val in out:\n",
    "        pitch_val = alpha*val  # ~ 0..alpha\n",
    "        semitones = snap_to_scale(pitch_val)\n",
    "        freq = semitone_to_freq(base_freq, semitones)\n",
    "\n",
    "        # simple wave\n",
    "        t = np.linspace(0, note_dur, int(samplerate*note_dur), 
endpoint=False)\n",
    "        wave = 0.3*np.sin(2.0*np.pi*freq*t)\n",
    "        sd.play(wave, samplerate=samplerate)\n",
    "        sd.wait()\n",
    "\n",
    "    print(\"Logistic map sonification done.\")\n",
    "\n",
    "    # quick plot\n",
    "    plt.figure(figsize=(5,4))\n",
    "    plt.plot(out, 'r.-')\n",
    "    plt.title(f\"Logistic map, r={r}\")\n",
    "    plt.xlabel(\"Step\")\n",
    "    plt.ylabel(\"x\")\n",
    "    plt.show()\n",
```

```
    "\n",
    "print(\"logistic_map_sonify ready.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "5d555783",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "### Quick Test\n",
    "We'll choose `r=3.8` or so to get some chaotic behavior."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "id": "17ac3c6a",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
```

"output_type": "stream",
"text": [
"Logistic map sonification done.\n"
]
},
{
"data": {
"image/png":
"iVBORw0KGgoAAAANSUhEUgAAAcoAAAGHCAYAAAAwWhJuAAAAOXRFWHRTb2Z0d2FyZQB
NYXRwbG90bGliIHZlcnNpb24zLjkuMiwgaHR0cHM6Ly9tYXRwbG90bGliLm9yZy8hTgP
ZAAAACXBIWXMAAA9hAAAPYQGoP6dpAABotklEQVR4nO2deXgUVbr/v51O0p2EJOxZTAh
hC2DUgaAQtiBImCiOjl6N+pNFg8LgMogrwyjIOAMug8XVg6BB9Oo4XAf06og6cWQVF8C
gsggoYFiykABJICl+/z+OJ5Ud3pJL7V2v5/n6ae7q6uqT1dX1fe8y3mPiTHGQBAEQRC
EWyK0bgBBEARB6BkSSoIgCILwAgklQRAEQXiBhJIgCIIgvEBCSRAEQRBeIKEkCIIgCC+
QUBIEQRCEF0goCYIgCMILJJQEQRAE4QUSSsKQrF69GiaTCt27FD1e8eNG4dx48b5tc3
evXuxcOFHDlyxOWz6dfv3rK0jXBlxowZyM7ORufOnRETE4oiUusGEIuSRKC4u9nubvUv34skn8S
bV1ZW4t5770WfPn0QExIwDjIwMFBUVoby8XOGWE3oiUusGEISRKC4u9nubvUv34sknn8S
4ceNcRPHxxx/H73//e5laR7Tn3LLzuPvuu9GvXuVXz9Yr
a47ZNTU0YO3YsTp8+jSeffBKDBw/G/v37sWDBAzyyySfYt28f4uPjVfw1hFaQUBKEHww
ePFjW/fX/YXt21fW/YUa58+fR0xMTDbv/32207vx48j/j4eMyePRtbt27F+PHj67ZcMyeZ
WHDx4EK+++iqKiooCbrvNnz66af47w9W9/G3DbCONArlcipNm6dSsmTJTjm6dSsmTJiA+Ph
4xMbGYuTIkfjjwww/drpebmbwur1YqLLroIjz/+OF599cv00Xp2nv/jU7XYTCYnoVn6kkXXoVO
nToTiP8fAgQPxhz/8AQB3Ed90000AgCuvvBImkwmkrVV68G4N71arfb8cIlLL+BXv/o
VYmJi0LzL/++1986ffp0dOrUCT/88AMmTTqhqEuLg4XKSkYMmSJSgtoZT8jEPk0HlrbaZ+eY/SsYYNkEqt
Hj0ZcXBwGDBiA119346wn7kydPY+bs2sRg8eDA6deqEhhtek8ZeOHIHZMI
zzzyDP//5z+jVqxesVgs0vUQmzuGDRuG//zP/7TvBaWfZTWZ9ymn5vtBpzcnHHd6PPHaJzXgHRDwjYB/TDbZzN7oqnBNm
5okePHgCAyEjvdkAQBWq1X2X2thH6hxKImTtZGtkTJk6ciEvRQ
lJSWwCwoLi7GtddeeAhYUAgO+++IvTJ05sE4/Y2Fie4+uLLePPNNzNz8jn/z4x+
YPXs27rvvvPjz33HD3HOIiIj2/AO+iL179wIArrnmGqzkZmAkgtFKGTWc5NSFG7YDLTaGpbMlOUzPbW2ZkNcgLxYkgzzNzEFUCuwbM3hgEgiUkFBATqTnhpRqzctXz37nM+AC8OFZWgdNoYYwZlVFadTrUX
6rF27Fo8++iijSO0jS0tLwwgsvYOCPwyACmk0w2osICAjDAONGOG4A4NSpUwCABQsWk6GWfPnsWsV77776ucePa
G4T//+Y9LB+HFF19ERkYGYli1bBrvdjrdvjme+eceQtQQEBRiz3j3J7wTv64Ljezzv1/l7NyUvHFxtVVo8/bnG06
jH5GZmYm4uDgqgMm1mgy8THUERCAiwrXC/39Ka68xEC6cTp6U8D5YRMcSA/PGzrdXa7eojgn1lY6aFPGbEjqtiaMtcyRmrAnzbL2hjDSWlCrmmdQWVJeb8xY
dJ2HAAKeDQ/+l9F6n67AJ3zRb99AVQZ1GpPdNKhQqFTUXvbMrPZY2mnPDzSeEE82eOTsq82R0KWvFmjZr0LuBDGYdfxyt
eRg/AByVtGnn8mGAEVmrXGQAAPUnPdhcNJCVbzVNiLaIOu94GAAqfSnrkaeVIj+JwqcCBCAt3U5EsNHvIvvPqvIaf23SQq4RAFlBlOaUN06pjcRUwx6L
ePw8Dtvh9uhlmHMDQhWNOTjPhcK/jBaxL86gxtsHPnQq9zBVtqcZl7Jbb5rODmM9TaDyGejkx4V0NrZx4G9lG7y9uTPvHWjQXbjGVJ
OXbHDq+XZJ0vqRdrZQ8hj7Kr9RgUk/L7sHc3hVcMsC8DtEEb5hoGnPj5PsOgO7svYOQsYRXyM6kTHKPKpO+LVvHt6NE66bfnhfvWbqdgORdFpvnsD
YxMnOKt+oEjtmsKsBBgTR+gN93p4NCNFIL9sJKhJfBzwW5xhCx0HzGiQG0b5P7TOa4cvfdyLyU7m54TDvPc3IJYyknXNG2+IC7TS+v9c/BjxhMLvLJFbD/x9NXXMr9G4jGsAWzhE+u8MT0KMDWPpOcwgwqy8RZ/3S6u4D5AqV3FnsL8KTnRfAvdBp1WUlqPO4mDLiQhoNRSrnbunR/xo2QJ3O+nE0nkxPOOXXW/ymR1Zhj0f7WG+ARq0Q+KxAnhGbwqW4gJbqeuaYCvdRELc0w4e0rg4dAG5BS8Vy6aH66YxxNy8JOxFY+jn8AhTFjUvf3vVD67jeyu+OPzhSbijkb2lcoWuGXvpkBTd1gmGFqufLMmlODKJ9tzIMnnALnHIWQ3K12tkK2xYEfGXFHLZkUMAh+6fdhZLnPXYJMiqtk0zn/0KOWzS7mbNKwMFOVYwrRZjfrcQMLqhCQ
Mhr7D8ZfpxHn0CxQzkSl7VKz4hTWRWEoXmj4pkMhJ5K7p9NXM+Xj2+CHPbq+4/M1DtN7i8+IfTaEaPxDJSHXOWZ
Ndw8vgNcpZoFhM1tXP85VhNTiy1u5K8zO0xgbT8w4PDmtCTi5gAk9mazWMRbqQcqzD9G41k3QlTYdGv/p3uc
0VBfXYptkfGT1+A3wC6GLXZ0oG2/G7qaEzONtHNfDA6cBHVNmHQqGDIqvbM0Bx8TfjfW7F6dZs1jt3Z11HfjfV4
eyPgZdpTHQ9ZnvDdkNyEstVEA3GVYsC5XibmdTDHkQY2tGKM4zVlLK2yMXRVz8AYMi0nzCaRz7btI6LtJj3PxDr2
vOhjfeE4VN+AcHGJ8kFgzIg6gtRc+3nj6vJqJUOWhgYm6L/ihW7i36Y9N4MUkyb+8fc/n6VezJJ2WdyZO5vGd+H7O7uOeyn7H14U5q+eG9GiqV7GfXmcS+GwXuddxiCqTZ61c7S6BG+AAQGg8rF95aq6nQOiV1Nt1i9Pxfy5w/w21bR
wnZKaP8dk6WO5QgPVDYj1ZyWjmHJAQnKfF7rGMGF5/JIiZ+ECKdTCsZFGkbC3CjAO/bR3Y6kDiMz6G1Hm5uEawd8SgA4jBqFCYKYXEKCWx9HFN5j6v4zkQmM5WPtR2lNmBrWzxKcwA5mSEE9H7Ne1sw+J
H3jZz5pO5lzA2f3gYA1XuWVzXxBEgsiFoJpQ3zn58bQaHj82lDXfr2SxTavqa8mDNnvVUKdUbmKc2Yf4LkfIlUKu2AG4cPh3EDVB6gjDx9+a+RA
rEGT7e5xyqKLwSk4ZuJSrr8B8ItKsDBTgK2aR3TnZhpbcb7tijCFlH9q3jgaLe3B8o6hAWeKIrT5B7PnGr1+FlBXVtLrX+4XTSjvH0OxYcoU5IN9+Yd
bGcO6Lc9Rz1f5i8NPz2K1ZK9+RGaGKZSo6FFeW3Xo0Q4kAwJQ+YsXDUxDTo+AMXIFymO7Dfkk4pAvXuvKn0ZYcoyOvo68pPYdtf3oRQ7Vz9PPCNDZ62Yk6D2C/uY7FV5OLz98MtEbj3KkwjqO8B6EwIyT5tpbxfTVdrqxnWPaBrk43zNJuIo6vSEf0NNPVTqiKJUX+vEiuDUPSOWhTKmF+zEe3bjlrVGmXrm0Jq4e3TtDYiv33d3YjgftUgPY8xxaTB3Bn2k4jqMidrv7Nb2Qr3MuBb4zVQnbW2MHoiLmLsqOMVCWm2wfNtTfNvQjTLBtLLmpvIZR/Vt2uvOe99tq1YRhm2Xq5F0NYJ8A3Fw9nBa2ekLjbsCfVGc3ZjWP5YfTHwKmzrCJFMFDWTIj3bydm9MbzYYjgn2q4FgmdSSXmo
v
4ni2Ypd
4eNcR

x2tragH4jYUzIoiRCknPnzuGrr77C7373O3Tq1KltudlsxpQpU/Doo49i//79GDhwIDZ
t2oTx48eje/fubetFRETg5ptvxsKFC71+zxVXXIEXX3wRt956K2655RaMGjXKaT/+8tF
HHwEA7rnnnoC2N5lMuPrqq9veR0ZGol+/foiMjMSQIUPalnft2hU9e/bEzz//7LT9yy+
/jJUrV2Lv3r1oampqWz5w4ECX77rhhhuc3I/x8fFt1rrrNZPoPZbPa7/ZdeeikGDBjgsnz
FihVoaGjocHt3x/6SSy7B27dmHJkiWYOHEiPvvsM8TGxnrcV0tLCwoL7F
792688soryMrKwuHDh/HUU0+1bd/eLUuUEjiSUREhy+vRpMMaQkpLi8llqaioAoLa2tu0
5KSnJZT13y9ozZcoUtLa24pVXXsGNN94Iu92Oyy+/vO1+/vO1m6i8nT56E2WxGcnKy39sCQGx
srEvsLDo6Gl27dnVZNzo6GhcuXGh7v3TpUJj44ESMFRo6Tdta44IOYNWsW/vSnP6P4+4wm814/PHHHsW/
fPpft3bUxOTkZzc3NSNHv2bEAi4u7/AoB+/fr57HptT1xcHIYNGwYAGDtDt2LIYPH44RI0Z
gxYoVeOCBBzzuq6SkBBBB999BG2b9/etv2YMWMwevRo903bF8uWLcOCBQt8+VmEwaFkHiI
k6dKlCyIiIlBRUeHy2YkTJwBI1ke3bt1QVVXlsl5lZaVP33XXXdg27ZtqKurw4cffgj
GGCZPnuxirflCjx49YLPZfP5uOXXzzzcTcxbtw4LF++HNdccw2GDx+OYcTk3LWxsrI
S0dHRTla8P5hMJrfL8a8P955Z4ffMWzYMEREREERODAgQNe19u1axfMZnNbXFn
Qp08fdOvwDbt37/b9hxGhoSSCEni4uIwfPhwrFu3DufPn29bbrfb8eababyItLa3NxZe
Xl4fPPvvMaRC63W78+/d3d3HO++84/b/e+XbXuQDDDUkUBYK47s9c
MJlNbWwXfffcdvvjiC7frr1u3zskibrwBOwjcXnLKdbkGOjImAVBIc2NbDFWxlsJjwSGQ
e5GW329GvXz+v66Wmpsms2o2Jms2H79u1Oyw8cOIDa2lqkpaUF85MIA0GuV8LQfP1Za+
++mosXrwYEydxVXXomHHnoI0dHRKC4uxu7ut7du/H222/XO6HZKDYl//jvm4ZcKs/KYg+wrsQ4jvp0bOhoE/wN6gAkTJmD+v/4U9
IiYnByy+/jHPnzgFw784784T3HXXXXXrCtjZpuSnmjsaBJT0/kFPqbH+RTxNmA6kDcjKgzhh4lnW6X2GtchyiQu8+cExq3tGeEYTZFfWcBU2YmkLHwHPfcS9VgfUI6wN
ARTm0HpGV+JMEQ/RXM2v6vSOxqXP0vOpehn4Gtgp7kWJmZ8N44n1+T1lQMH6U/J2SNotZR
0dNFOGL8dlTMA54LwGJmLkjNyLFCeuq89vfkpuIsNNdqHfAdIRKVv3PvDdxAI8aIdMJvl1lOkBzJx59sSWwpc6QUVFmOabAbCDGHiYMMRnbGknK5qQ4GWjvwwGMixMQOyv6Ld
RrvAEV2IM+Fi91zFB/s9e6Azb2QiDhm/mDDa1D4+xUB7ZlUQW9Dx77XkYAkcZO99YAvZNQCzvY4/pD6f5QCaV5thvwgLFWPDx3dqyNLYdNUQQaWfJZ/KbHzaP/UUoaLQ
Mx99AlOLtAoM/AnqhSGl0vUEnmY3GFzD5hxoL5Ga8HbZ3OyvOmfbg5FA1eVBXg2bu65hJ
95qMpF3o6eXdqa0RUQ8SqNLPb3vrH9BsBi2iBPtWLjrY/A8ZLIuMS8wb5u7pADeurOhvJH
/o6DgKVGGZzk4LzTdNAPj3ZlKLiwUcEkjOmTNuDmYYHxNmKjR9Wi5qE3dzoRhxIsIKmIIJ
9iZCdWwvL//qDbbzJtLLG9k2SpgHPP6bDUukbIJd46GXANnUvAcNsSZ1nUctqD+xD/tk0/pY
HGm3kkaRRgK3B9/ef6GaL+Xc2xHfwz/+P4Ib5thSgBHGr6mOKyLR3JT1Bh8gnGL8tVfo/cA
dUZDAcCxHIu2I2EVeqeeZMzTRs6gFw5x+uJWfBItDPbg3uHGgl/cQnKsUfF6FE9r+GoDzYeK
bAw4P7Z00CRbzdbyQA8AgH8OYixFOg6dMHjRR4NtmWbXbvnRCMGf3+Z8/bsw4D8KPddu54Mj
eb66Qg2tIQ2jdmY8bBfiJg3eL2lWVAAMZOvRoI+ZGsZBqy+AfU6W1yHaSOGnXoyeYYyGX4o6
02T5Cf5wMRnhwnOAMXY2AGZ1LmQ0DrIDhYH5DSfsbUzd5zZ5A1mnTR35C3PHA0Gy2Bxet2Ma
qAVqXO9XIG5aEu3+tHLJP09x/6hEnvVPqPcg51XU5P5+M91ovS5Cc7lI0dXt2qyJ4CFOkO4W
P2TxIRGF5iKaRmhwffkJj0npVDa+hKU5eU+cvE+cuNqcvDt9OeIiNwQzr7fILmW1e9lyEF
uV82eJUrj3G1uPOfbZP7kdHEAt94ADqZ1t4wpVw2o5BAqh8gXKhO7/1uHvK7/Xfp1h+GmrSE
dh3YTVGa7JAGO8PjDwfSBuWwSjbmsc0m9tw2F8HRJJH8T7p/JmWRV4EQ0YwSe+UQv2tbBJD
Kdv8HXD1ji5P4tDrrEjPvhhxU8Y44XwqkBh0vRXmLqbmpJxZvo1WLv8dCydjIKrcXA3dbEOm
/ktiwq9Tk/OcKjmbe8Qtdlbk5gl8Ui0KDAhcdaaJR0Nc7kt1GCttx1tPFH2Hds6UrHHZl/Ey
ebFt6FHM9n2QjLDG4XqJqEXo0YkAwPDdvedTU7hzfl8M3onm/cE1ZI9mo08bnVfUbAGk8YZL
iqW1aUuE3g/zGFO7j8+Kh9UFoCp9e00cchIkHmBAZTg9H2M2BvTaUxUp6IK9mcFfF8gjUKFd
eMkpp0wzcP6bABcW9xiORdGiIOjTPIcAkFy1b9d/DUFnUq7nJOd4yBmHV3ZYJHJxiXeXTvXj
p3sG5lgTbJAlEtmBiUuP37sIYgB8NVHm9zV4VVX7w9a/eX/jJP8Jxx2u3z+zaBXeJUyiL8nGW
8N5WgvuYyvKJS6ryoDYBHxPYXa+IDjfsjOhPuTI4yfFDVuiqQlqmJc0EIdBfiHrzyLKoi7WOW
c6H44yhjeWcHJrzc3BI42HAoiQ9dswAVw7QlYrxL7eLGtxqjIjv3pbaHqJ1kwvbsgAtzr8W
0iE3Fct9Pmx6Ts41NRozDZr0xJIgJEzjdsyXM8DlWZkJtsybGWVeMcUuwGhEvVUv8dmFkOgW
8sOnufd6gaGvH47Zc9IzNbSqaTt9ZhMSE+T3KMbHqV7RCw1f+xzzp1e2VzgMG8/fcS9/IB/C
oMyz/zTm4Xz8Fk8rdJZOFMT7IHknYL8u8tbqdHYsh1+OLTjX85SKnJa8ITx2Qy2YY0pwfDlb
K9qTLQ8H5cORBT1kuyOTjOEEVKFpvvsvJINtD7ISPo8wJL1GFUSmpz/7lF9oqnC91GRoHncQ
TGYqUxgWRi7KbvGKfZhBrZBj9e7pjXJf5tchaq1j+hO2tDpaDHuY6xfJ8GeSLAaVHvMVN7ZD
7hIs+Ghz/VaxEnA/jiBOL8PkdrT1Q2w5nmyYc+f0JvIHYvuJ6hMRCa5PnKFGS8rmOPUDuOR9
q7xmyf/h9gkDKybD+FSvAjhAhX5LdB2NUdQ2CG6Yd1X7KK0dO8t/fDtExMtvgfh/Bh+X/f9
IOiPXdQTvs2wYkVGb06VTLMWN+PW4awxk7hMWXAq77Qv2EW2v5fn2yEiV3VHVKbr7MVyB+uX
QSVIqqqIHDFPbvf7fX5Rm6KSMlXDE6X8g9MwRk8tt/+dHcdqgt3OrC0PPvK9Ye3txpPlZ2DG
GJCUmVQlS1H2hxsOnp/DdfZyGhpkW6cApsTn4iuPOx3TiRAJxNnH7BtqlH0sc9QmIHAnDRS0
Rm3dG9KrsMrbTE3/7x/9b//3fMbNAW3/cUaAkIHhmXTyvlmn+OEK4d44kO/7kt+sTYdGlnaZ
K5FS3YM2SuA8j+UPYS9I8vvtqGQpPE1DAIH1ft4U6Fx7xUlm2VZiyfnPGHvZhSrRgJh2EBEY
zY2xUgjHIjxwYLgHx8X+gIAQ3tB8/h2Tb0O3kdkYnP+u2EGKuK6OvgGwFuGjV5Sw1PK7L8gM
cHoTdRoeg2H1nG4cf+u2EMqVvpnmq6P59VNeSOd1u2hXv7W5/MCqwoDvP3e3WlJ2UaQgyeJ
yHRaGSZhn0lNXK2hxmR4QxRZg2b1Rvgd+Aw+q0xMqSRlm2NO+NLGj4Lmw/vtOZX4nB51W4
vxJG64dNUhZpqcW+j2F5C5Kt5ub8eqBTzQCuSyQ5fyJaY6egCJn2Qz+B3mmqs4KjzX2kqk3M
6+LuJn+EnH2DRB8LQSPk+UJyTeZPyb6Q6HsmGeq5ASxQTGtTuFyj3AKuXeXq/YH4EvOQN6KR
6pZFuTlAMqT2Io4nw2K8CWAIewCCkzJBqnn0v4wDzzFZsT5hN6zTo4wAzGnz3wXxrb5Fx6Y1
ZdMUjkQgyZzwqIi/DpCH5iJtwu/GIhfJKi6lJWnN+Z1NRK/N0/kfO0aZOMhzHs3kJrN71
WDpwPhqPp1+gRVRjkX0GeFBl4NXUovLCLhBXg9ZPvlZWVVj/JFqbTGOw4L7GR11uKkNxnCaZ
t/BoM2E5chSC8AfR5bQhC4OcBD3eT5WtWjFYQDAzwc8OBpqCmsM6vYmMGzjk0AAzjcUyS+QOK
y7SAWxBo+cHv3Z2Q7qUbyvhS+sBaUg0aW0q2n9fvbKWL3aLBq35xPZTQ9/4nl9C9x7SO+nRa
KcE8LDt40z4Y3g6cpFv8dgf9l89Tvfd5FmvDzMABTd2SRTnxIGh6GXTY1G64mc+DxvWI45sF
hB+U0NPf/vuG3JP0z5tcTWWHXe/u6VBNAgICAgICAgICBOiQfAEKYaEYVRXXOcPXLRcTMDKMw
rABO6m5kAnoYTq1w6w2ilsN/EVsgmf8y7/NEhGW3+EWdDlyEJcMbTbpIAI+/kBkBVWKAhR
XLGhdxEQ7vbhCjDE9Tgsg5M+w+pcy0u71rqJEZjp2TmwXoxbqgZBe2JzmRtUSfRwfYkjbpOS
7qkVKNfOg6upfiM7D3DFw9Eeo3/sx5XDyFZdLxHlumjBGzKjZvS8BrM/16pyM0OrGgZtgU1n

GSq8/+ogKKhgFcpv7jgaWt+ZCWVxcjMzMTFitVuTk5GDLli1e13/ppZcwaNAgxMTEICs
rC2+88YZKLdU55LbpGGFRZmQAXbrw16EmlAcOSK/NZu3aQfgH1SH2DY0sb02Fcs2aNZg
zZw7mz5+PsrIyjBkzBgUFBSh3zEx0YPny5Zg3bx4WLlyIPXv24Mknn8Q999yDDz74QOW
W6wxy2/iGsChDVSjtdmehrKnRri2Ef6SlAfffL72PiCC3uTs0srw1FcqlS5eiqKgIM2b
MwKBBg7Bs2TKkp6dj+fLlbtf/n//5H8ycOROFhYXo06cPbrnlFhQVFeHpp59WueU6g9w
2viGEslev0BTK48edk5VIKI1FRob0esECcpu7QyPLWzOhbgG5uxs6dO5Gfn++0PD8/H
2bXO7TVNTE6xWq9OymJgYP3112jxkL3Y1NSE+vp6p6p0fIQW4b33B0vXbuzF+HklA6WpM
ACaXR2LdPeh1K2dhykpYGLFwovVcpYU0zoaypqYHNZkNSUpLT8qSkXj
11Vexc+dOMMawY8cOrFq1Ci0tLajxcFNYvHgxEhMT2x7p6emy/xbNSUsDpk6V3pPbxj3
uXK9nzmjWHNnZv9/5fW2t0gAuOHH6TXJ09q1w69M2gQfx44EDhyRBXLW/NkHlO7cV+
MMZdlgscffxwFBQUYMWIEoqKicN1112H69ahrq6u7XH06FFZ268bunaOkAALOHxIV58+ahrq6u7XH06FFZ268buna
VXs+fT26b9jQ2ShZWqLpehUVpsfBnsiiNhaNFSULpmT17+POIEaoZA5oJZffu3WE2m12
sx+rqahcrUxATE4NVq1ahsbERR44cQXl5OXr37o34+Hh0797d7TYWiwUJCQlOQlQlOj5BEnDw
A0NysXTv0inC7JiRwt2soC+WwYfyZhNI41NY6iyMJpWf27uXPgwer9pWaCWV0dDRycnJJ
QWlrqtLy0tBQjR470um1UVBTS0tJgNpvxj3/8A5MnT0ZE+xhduOEolNXV2rVDrzgm8gC
hKZTC9TpqFH8moTQjYkQELpDSGUF1+s2ldGqvZNbpg7dy6mTJmCYcOGITc3FytXrkR
5eTlmzZoFgLtNjx8/3lYreeDAAWzfvh1nn55nkY26nZpnz3Uo2wgtzw9dg/Nl+kwxNv6+VF8rjKKUCg
oCAmlK46JPEDoCWVTE4/XAIDoaJJGTQGchlBddxLOXS5jd09IidQhV9FcrCwkLU1kF5ZTDQcL86
i0aJFqKioQHZ2NtavX49evXoBACoqKlBeXt62vs1mw7PPPouSkhJERUVhzJgx2LJlC3r37q2WTvVTeWRaC3WF3FwqG43gNDmyOFAgboVqZaMiMhjsWk2t
a/QKdIHpYAhJKVxwTeYDQE8pDh/gQoU6dgOxsvoyE0jiIRJ68PODvfwdOneJDvKhohDM
//cTFMjZW8g6pgKZCCQCzZ8/G7Nmz3X62o/X62evVqp/eDBg1CWVmZCq0yGMLt2rkzz+xFn
l1F2vopedlQWIeH1jIx9XGROjXbsI3xAW5ahRXCgZ43FLx7rEhGQUDBrkOiROQcI8sBc
iCKHMy+PP1dX8QiQt31KhJ5BgzgCUuRv/SBaYiIMRBCmZ0tZbCT+9UVDeKTAAllaCC
E8sor+fP583wKKULCk0XZ0AC0tmrTJjlxFEqTSbIqyf2qfxobpobpfNz0CCgRw/+moTSFXG
vUzE+CZBQhgbi5LniCmn2CHK/SthsUrKTsChFZR4gNIoOLpeARJKI7F/P/cAdevGRZK
E0jMaDA0BSCiNz5kzwIkT/PXgwJRUBKCl8WFcUTX4DQcL86WpQACaW
REIk8ouIMCaV7WlulDiG5Xgm/ENZkWhqQmEhC6Q7h1kpLc04ACJU4pWMCV//+/JmE0ji
I+CQJpXcOHeLDoGJinAvIqwAJpdERQil6WCSUrrRP5BGEilAKazIlRZQmoTSOJBQ+oZ
wuw4cqPqwGRJKo0NC2THtx1kj1AKQk0ohdsV4PEugITSCAihHDiQNL4kFB
2jLAo2w9QDmWhFBYYlDQ/RN62t0v9HFqV3NErkAUgojQ8JZceEukXZXX
SDAmlezQaGgKQUBqb2lpAzL4iTh4SSlfaj6EUhHIbSKfj6EUhIpQerMoSSj1jXC7D5QybKlK7YbFJ
2MAkl4Reih9WrFxAfz1+TUDrWGn8zBGQmlk2ifyAJjAKfQ2R7oulRFpFVQZAFn1g8y8HKh1Fa27Q7
/noTSyLR3uwIkl05fRo4e5a/Tk93/iVV7ZxWx2voE4CiWVM9Qv7RN5AOoAOvPjl9T8AM1R/s9l
CoxiGHGiY8QqQUBobb0J58iS/0MId4Xbt2dO1HgoCKWwJvv04UUBCLr9cIFLqSEPnF
nUVos0jAfcr9yNIxPAiSUxkacPGJaJUDqjdrfKqecMeT2xWQytiFglA6ul0BXnUoOpq
/psxXfcKYa1UeAcUpndFwaAhAQmls3FmUUVHS7APkvfWcyANIFqWR3VvuMl4BKoxuBCo
qgPp67koUFZUEJJTOaDg0BCChNC4nT0oXf0oXUfveqJ7ilMeOARs2SEXJ1cabRl4KrtF2FiV
AQql3hNu1b1/J+heQUErY7dKxq6SqS5Ei5Q48fz55IS9dvgaQw
lIAllXZ1x47kklMbFXSKPpgIRS4uefeZw9Opp3KjSAhNKouHO7vCvQglMeOAXffLaW32+3
AzJnqW5a+uF4BLpZGo7mZD1gHXF2vAAml3nGXycXCMgZZQQbtesLGlClCcpUhoTQ3oQyKYk
/aymUBw+6jgGz2YAff1S3Hd5cr9S/HR0vydRnS/HjrEj2lcnDR9mCMklPqGhNI3NI5PAiS
UxkXvFmX//s5TWgE8aFAFjxykd3a4mk+vnVBhd33jKeAWk8Tg4
JpeZDQwASSmPCmP6FvL9SvfFXfcmE7BiBV+uFkeP8ue4Tk2TKoyuD1GFkpPGa8CKoyuX+r

qeNYrQDHKjtB4aAhAQmlMqqv5zc9kcn+R6UEoAWDMGOl1//5AUZG63++YyOPO4gKMLZT
eEnkAcr3qGeF2TU2Vigs4QkLJYYxcr0SACGuyTx8pxuaIXoRS9JgBaZYENfGWyCMgoSS
0wFt8EnAWynAuQVheDpw7x5N41AzbtIOE0oh4c7sC+hTKlhbgp5/U/X5viTwCIwulcL2
SUBoPX4WyuRloaFCnTXpEWJMDBjiXaFQZEkoj4qtQ1tfzhBatcBRKQDrp1cLbGEqBUYW
yrk5KVPJFKMPZKtEj3hJ5AB5XF7WJw9n9qoP4JEBCaUw6EsrERKn3peVFduKE83u1hVJ
YlKHoej14kD8nJfH/2x0i67W5WZpBhdAHHVmUAMUpAV3EJwEdCGVxcTEyMzNhtVqRk5O
DLVu2eF3/rbfewmWXXYbY2FikpKTgjjvuQG04ZfV1lPEK8MQVPbhf22f1kUUpHx25XQE
ev7Za+etwukZ8Ravyihcu8DGwgPtkPAEJpS6GhgAaC+WaNWswZ84czJ8/H2VlZRgzZgw
KCgpQLiyBdmzduhVTp05FUVER9uzZg3feeQfbt2/HjBkzVG65hlRW8pt6RIT3i0xPQnn
VVfxZTaG026XhIaFoUYpEHk9DQwAqjO4NLcsrimIciYlAcrLn9fQilFp1KBwzXsPZ9bp
06VIUFRVhxowZGDRoEJYtW4b09HQsX77c7c7fpffvklevfujfvvvvx+ZmZkYPXo0Zs6ciR0
7dqjccg0RPay+fSVrwR16EsoJE/jzDz+oV1O1spInEJYXSm8WJUBC6Q6tyys
6ul09DVsC9CGUWnYojh/niUzuZldRGc2Esrm5GTt37kR+fr7T8vz8fBQXF7tdJy8vD8eP
xY8ewfv16MMZQVVWFf/7zn7jmmmm7s8fk9TUxPq6+udHoZm927+3FEPS2uhbG6Wbs65uTw
xoalJqk2qNMLtmpbmvT6kUYXSF9crQELpDq3LK3aUyCPQWii17lAIo6B/f9fZVRGM6G
sqamBzWamBzWZDkqhL+gtJSUmorKx0u83VSPx1ltvobuvsZTOgzT1tNiNHjsSw3eUvxl0aOCTvPawvUyfNp4Yx7b
XIzExse2Rnp4u6+9QnY55noFSCjd4a4//DlIjrlFX1J5AG0F0qtOxQ6Q6SeQOUvoyhBIwplIz55noFSCjd4a4
6lJrlFX1J5AG0F0qtOxQ6SeQBdJDMY2rnemCMuSwT7N27F/fffz+eeOIJ7Ny5Ex9//DE
OHz6MWbNmedz/vHnzUFdX1/Y4fvx4IJ7vH4QU20WoubHZDV5d+tnVJa4sKU2yPaqjI4+XPuQOPeGsFPhT+4tPuvSrqTPcNpPX9WMMyTQE6oqrrbSpmA3yyfiQ8QjefcY0kCAcxRbGR4XX+I
eTGB+vQnhcR0+rPP/a4iwrrs4BxG1UhnRY36aGxvOwpwtSqnsR49pJKWQq7ojH0+TBtI1tkGr4WVSeMBzaPxQJTwWw/dVx6dw68dbqu4eRGN5OEUeeEl9qNCMuJ6VwdJ0X9eO
2m12sx+rqahcrU7B48WKMGjUKDz/8MC699FJMmjrQjMYJUKDz/8MC699FJMmjQJxcXFWLVqFSraj9n7BYvFgoSEBKe
HYfEl41WgJ6EE1BdKfy1Kxowz1ZZwu2ZmduySIovSmdZW4Lnn+Ovf/15afs5z/1afsMN6ny/zSb
9f75alFr9d2lpwF/+4rzshRfU61CQRQlER0cjJycxcuRIt9s0NjYiooltLS0sxcuRIt9s0NjYiol0
Px2w2A+CWaMhz/DgvImA2dxybCneh9NWitFikgd1Gcb/66nYFqDB6e955BzhyhIvQ4sV
Sksjnn6vz/T//zIeHWCy8o+MNrS1KwLleM9DxfUcuHI2CcBZKAJg7dy5effVVrFq1Cvv
27cMDDzyA8vLyNlfqvHnzMHXq1Lb9b7/9NqHnnKgDWHHnaDbHKNJlFMJHmkAllVVFirPmzeSw9MVXW9FW
vmY2hgmNw22Lxvq5ehXLfPnVcnL6MoRR07syfz5xRqjXXy4mvGK0AWpSOMAU8/zV/ffz/
vIOXl8febNqnTBpHIM2AA7/B6Qw9C2b5oyGefqfO9FRXcwxMR4VuHUGG0mS76FwoLC1F
bW4tFixahoqICgCaGmT4cFIamwAE1oxAQ6sNPj9nNDeUuTqeDjfQxL/e6ijnQa9OKLL+LBBx9E586dMX78eDw
tTv5Qx1e3K+A8eTNj3tPQlUBcYEIo+/ThbsLVmg8VX1yyvvA3a8VFcaxCKQBa4D6rDEyE5dHgY69UVg32Z12uBrIg8gCeW5c8D585LnQ01Ehzcqig+32rB
Bne8Vnqd+/To2ClRAU6EEgNmzZ2020GnbsXr1apdl9913H6677Z6FW6VT/BFKcZG1tPC
embCa1EJcYMLSj4zkPcPvv+cXgZJCWVfHXdSA70IIGEcoA3G9klBK0zkwqLcud
OPmYvPl7ZNvgjlAkJkkCdPOnbuSw3osNbUAC8/z6wfbs6x0lHbldAB1mvhB/4I5RWqzT
PnRbu1/auV0C9OKWwJrt149ZDRxhJKFtapPBNvwoPEvw2CxEMMjJBBkkrPqPaDtVZA0XnTGhM
et9mAL75Qvh2+ZrwC3AuktftVCGVuLtC7N0+G2rpV+e/VUcYrQEJpHAIp56RlnFJLofQ
1kUdgJKE8dIjf1GNjvVccEojhIa2tkpUdjjzzDH++9VZqTEoxYlUq7Xxnz6IEtBdKR8/
QlVfy12q4X3WU8QqQUBqHo0e5yMvdGyQ0yvWVsM0VdyTlZHhEvoGtT7ZWIyZkGkBtw
7XDNff/wRWLuWv37kv9sxkyfQPFsKpm44P6sNgnZq9+ZWkHaeKp5Hs87fej9VgqxtgqRWF8Qh
ojYM4cQYM8L2ck1ZCWV3NM1sjsCaaQbOJ9EhsC4Quk4R6nfO45fj5ec437Qh3
Covz6a540oxQi4zUz0/fEHL0IZUqKZFF+842yGeJVVdLEDzrIeAVIKI2DP/FJgVCKVdCvZWZWVBlbo+Ps82hgp5wwRN8rpQ9rQ
1SUnOKfD9+nGL+OxZZetFhrLr1Z+MV0E4C2VVFSCSAt1ZkwCfYCAlhdcn/uor5drir9s
V0FYoL1yQronUVOCii7g3y25X1k0tPE59+miT6esGEkqjYEShdHdAES7AtwSFm5jJd2v4eB

69aenHc5C+d//zYvxjxjhOnheYDKpE6f0J5FHoKVQiuvYapWy5tVwv+osPgmQUBqHUBB
KQJ04JblendGDUGoxp2FDA1BczF8/+qj3scRqxCmNZlE6ul3FsVMjoUdn8UmAhNIY2O2
BpUuHo1A2NUnfH2oWZX299Nv8EUqtC6OXlPBOi9pzGq5cyWNpWVnAb37jfV1hUX7xBXf
BKoHRhLL9WGgAGDeOP3/7rXLnk86GhgAklMagvJxX54iK8q9yv1ZC6Zgp1x6lhVJYLDE
xkiXVEUYRyoMH+XPPnv4VkNDSohRzGorkLbXmNGxuBp5/nr9++OGOM4QHDeLH6fx5PuZ
SbhoapN9sFKF0dx0nJUnXsFLWN7leiYAQroisLC6WvqJ3i1KJzFeRyNOrl+9l+4RQnjm
j76m2AnG7AtoWRtdqTsO//51PIpCSAtx+e8frm0xSDFOJOKVIwkpKks43X9CDULa/jpW
MU548yTt0JpN/sVyFIaE0AoHEJwFJKGtr+dhGtfAmlGL835kz0uTOcuJvIg8g3bjsdt7
z1yuBZLwC2lqU/fu777A4DhuSG7tdKjAwZ47vtUKVLJAeSCIPIAllXZ1yLmFPuHO9Asr
GKcW9rndvafyvDiChNAK7d/Nnf4Wya1fJ5aTmTdKbUFqtkvtYCferv4k8AHfTipupnt2
vgWS8AtoKZVqae4vusceUs94//JALU0ICd/P6ikjo+fxz+TuWgcQnAd6JE0Os1P7/PIV
Q8vJ452fvXvk7uzqMTwIklMZA9LLcDZb2htks3SSrquRtkyfsdunicSeUgLJxSn/HUAq
MEKcM1vWqVTKPKIA/eTLwwQe8U/LBB8CiRcp8nyh+/rvfAYmJvm936aV8/YYGYNcueds
UqFBGREjJWGq7Xz25Xrt1Ay67jL/euFHe79RhfBIgodQ/drt0kQXSy1I7TllTI/XGk5P
dr6OkUAZiUQL6F0rGgne91tZqE4MVN9zLL+diuWIFf//kk8D//Z+83/X55/wRHQ38/vf
+bWs2A6NH89dyxylFVR5/hRLQLk7pyfUKKOd+1eHQEICEUv8cPswz8SwWXkHEX9QWSnF
x9ejhOfGILEr/qazkFY0iIvw/D4RFYrPxWJfaHD/On8UNd9o0QEyVN2WKJCJyIKzJqVM
9ezS8oUThgZYWKXnJKEJ5/rxzVZ72CKGUeyJncr0SASF6WAMHdjwjuju0EkpvNymlhNJ
uD12hFG7X3r39n8jWYgE6deKvtch8FRblRRdJy/76Vy5KDQ3AddfJI+B793KXrskEPPR
QYPsQccotW+Szvn/8kXtZOnVyPga+ooVQOlblcee+HjuWd9p+/FG+oT41NdJ9SkcZrwA
Jpf4JNONVkJTEn/UklFlZ/GbmeGHIQXU1LzgQEeH/DUnvQhmo21WgZZxSWJSO/0lUFPC
//wukp/NOwO23By9Mzz7Ln6+/PvBi2kOH8jlMT52Srr1gccx49XXIkiNaCKVjIo+7Nic
mAjk5/LVc7ldxnDIypI6dTiCh1DvBCqUeLcrYWCnBQ06rUliTqan+jTcFnMdS6pFAM14
FWgllU5P0ne1deD17AuvWcYv3X//iMctAOXYMeOst/vrRRwPfT1QUMHIkfy3XMJFAE3k
EWlqU3uY8ldv9qtP4JEBCqX+MJpTeqvI4ooT7NZAxlAJR6UavFmWgGa8CrYRS3HCjo6V
YqSPDhvFScwDPgn3vvcC+Z9kyHgvMywOGDw9sHwK545TBJPIA2luUnpA7oUen8UmAhFL
f2GzSRWYUofTFogSUEUrHqjz+Qq5XZejIhQfwxJv77+evp0yRLDBfOX1ayqQNxpoUOBZ
Il6N6lBEtSk9DQxwZPZpPm/fzzzzpMFh0OjQEIKHUN4cO8TnhrFbJVekv4SSUwViUehb
KlhZ+LgCBu161KozuLj7pjuee45bc2bM8xuhPcs/y5Xy7Sy4Bfv3rgJvaxhVXcHdwdbV
kyQeK3S51dgNNUNGr67VTJ36sAHmsSnK9EgEhTpxBgwLLeAXCUyhDzaI8coRnTcbEBJY
1CejDovRGoMk9Fy4Af/sbf/3II4Ely7THYuHzVwLBxymPHeMTGkRGBja8C9Cv6xWQL05
56pRUqISEkvCLYOOTgCSUjY38glUSxnwXStG7rqqSb8hCoENDAH0LpXC79u/f8SwYntC
qMLqvFiXAz9V33+UelH/9C1i4sONtXn+ddwJ79QIKC4NqqhNyxSmF27V/f/8TzARCKE+
d4uEYNfDF9Qo4xymDcVOL45SeDsTHB74fhSCh1DNyCGVcHLdEAOWtytOneZYj0PEFFh8
vWX7+xqQ8Eaqu12AzXgHtLMr2xQY6IidHSu7505+4cHrCZuMuWwCYOzdwIXKHXHHKYBN
5AMltzph6HR1fXK8AzxCOjubCKqaBCwQdxycBEkp9I4dQmkzquV/FxdWlC7cKOkJO92t
DgyRywbpelZj+KxiCzXgFtHe9+uMynjJFKj83darn82PdOj7gvWtXYMaM4NrZntxc7i4
9doy7vgMl2EQegLeja1f+Wg33a2OjNEyqI6GMieHHCgjO/arj+CRAQqlfWluDz3gVqC2
UvpYOk1Mohdu1S5fAXDdCKG02nhiiJ4wslP5alIJnnwXGjZOSe9qPb2VMKld3773ccyI
nsbG8Ni0QXJwy0Om12qNmnFJcxzExfAaWjpBjmIiOh4YAOhDK4uJiZGZmwmq1IicnB1u
2bPG47vTp02EymVweF+v04AbFTz/x+ediY3nZsmAIB6EMJpEH4MdZuO705n4VMcpgXK/
CfadmnIuxwCxKQEru6dWLu/TaJ/ds2ADs3Mlv5vfeK1+bHRHu12DilHJYlIA2QultSI8
jYiLnjRsD98aQ69Uza9aswZw5czB//nyUlZVhzJgxKCgoQLmwDtrxt7/9DRUFW2Po0e

PomvXrrjppptUbrkKOGa8BprAIQgHoQwmkQfgNwQ9xinPnpXEpn//wPcjhNJuV6/6UH2
9lEDmr0UJcHEQyT0ffggsWCB9JqzJO++URERugk3oqa2VhM1IFqWvGa+CK67gHZbq6sC
u5TNnJM9DsB0KhdBUKJcuXYqioiLMmDEDgwYNwrJly5Ceno7ly5e7XT8xMRHJycltjx0
7duD06dO44447VG65CsgRnxSoJZT+XmDiojh+PPii2MEk8gj0KJQiQaJ7dylOFQjR0ZI
bTa2EEHE+JCYG7hodOhR45RX++qmneFxy1y7g3//mHcgHH5SlqW4ZNYp/x08/STdyfxD
WZK9ewbuGtRBKXzu8Fgs/VkBgcUpxnC66SKqQpTM0E8rm5mbs3LkT+fn5Tsvz8/Oxbds
2n/ZRUlKCq666Chlebo693ehgCIwqlvxZl586SqAab+Rqs6xXQp1DK4XYVqB2
nDDQ+2Z7bbwfmzOGvp03jEzIDfG7LQAtx+EJCAjBkCH8diFUpR8arQCvXq68I92sgcUq
du10BDYWypqYGNpsNSWJ2i19ISkpCpRh46oWKigp89NFHmNFBttvixYuRmJjY9khPTw+
q3aqxezd/zs4Ofl9CKKuqgt+XN/wVSkA+92uwrldAn0IpRyKPQG2hDDQ+6Y5nn+VJI2f
PAl9+yZf9619ASUnw+/aG4zARf5ErPgno2/UKSAk9Gzf6PwsMCWXHmNoFixljLsvcsXr
1anTu3BnXX3+91/XmzZuHurq6tsfRo0eDaa46tLRIN8hQtigB6fcFK5ShalEaWSj9KTb
QEZGRwPPPOy+z24GZM+WbD9EdwcQp5cp4BfTtegX4+NdOnfi18+23/nN4zhdBMKLt
37w6z2exiPVZXV7tYme1hjGHVqlWYMmUKoqOjva5nsViQkBWSj9KTb
NofSnKo8jcliULS3SxR1qFqWRXa+BWCbeOHXKdZnNxsdSKsXo0fx53z7/rx+jWpSBuF6
jooAxY/hrf92vZFF6Jjo6Gjk5OSgtLXVaXlpaipFiPjgPbNq0CT/++COKioqUbJ2OA6
+laN2pRDKkyflm7W9PQ0NUoaj2kJ5/Dj/XRaL9FsDQW9CyZi8FqXahdHltCgB9yX8zGa
gXz959u+Obt14sXUA8DJ0zYXGRsnLYTShDLSDE0icsr4eEF4+nWa8Ahq7XufOnYtXX30
Vq1atwr59+/DAAw+gvLwcs2bNASDdplOnTnXZrqSkHa7+16
5HIheaHy8f7OTi4vj558DH+gvbkjp6cENpdGbUFZX8xuJyRR4QW1HjG5RpqXx8nZiggC
zmU+vlZYmz/49EUic8sAB3tHp1k2e4StiHzU1ynV2AS7wIgPdnw4vIMUpN23iBVN8QVj
dycnBZXUrjKZCWVhYiGXLlmHRokX41+hc2bN2P9+vVtWawau0LU
mAfmFMipKOgmVcr8G4nYF+I1EuNpFlqC/yJHIA+hPKIXbtXdv30oCdoTahdHltigBoKi
Il5TbsIE/q3EfCCROKafbFZD+O5tN2XGw4jqOjfWtKo8jv/oVz2RvaAC++ca3bXRekUc
QqXUDZs+ejdmzZ7v9bPXq1S7LEhMTUXfx0lCxuKF0NNaq0xhH8fOaWcO6nqXi2cHZ2bk
kqmqqXUo59ZCXCK8uD1mEP9MDv6CPPRUXYOUjjMM4TdXnkPjHxmbJtJFDXn9bVjzqr
9hCdRaN9ZTS9PAGAB4FRt8GCyCtMS3zGF6ThuNPOEuL1yKvGCp+8TtFKhY1aXt9bvOGJ
3Dxvm//ZyjKEE9CeUcrpdAXUtSrs9sFiXL6SlKW9FOiJib999x88NcZ54Q85EHoCHFRI
SuIfh5EnlrC9fJtr2hNnMOxX/93+8IyPmqvSGAeKTgA6yXol2NDdLg8zlFkpAfxYlEHy
cUo6MV0C6AapVuaYjjCyU1dXc+jGZuFvNyCQn82QqxoCtW33bRm6LElAnThlIxqsj/tZ
9JaEkAuLAAe7fT0iQt9estFAGE48KVihD3fUqR8YroK5QivMhKYkP7TA6/tZ9NbpQBuo
FEEK5ZQvv9HeEAYaGACSU+kPujFcBWZQdo7eptuS2KEXW6+nTvidbBIoS8UktEXFKXxJ
6Wlslr5DRhDJYd3l2Nj/PGhuB7du9r3v2rHTtkkVJ+IVSPSwjCOXhw/wC8wfG5LMoRZ3
Jlhb/2yE3ra28xiggn1CKuBZjylvNcpWv0wvCovzmG56s4o3Dh7k1FRMjzzhogRFcrxE
RvrtfRfJez55SJ06nkFDqjXAUyh49+IXCmORu9JWaGuD8eW59B+uq7tRJGnqgtfv1yBE
u2FYrH/YiB1FRUmdAafernOXr9EB6Os8+ttmAjmpRC7drVlbwM/84YgTXKyAJZUcF0nU
+WbMjJJR6IxyF0mQK3P0qXDfJyTwzMBj0NNWWcLu6G2QfDGoNEQk1ixLwfZiIEvFJwBi
uV0ASym3bgAsXPK9nkKEhAAmlvmhqkspxGUkoHQcpB3qBBSqUcrldBXoTSrncrgK1Enp
CzaIEfC88YGShDNb1CvAhMcnJ/H4mCti7wyAZrwAJpb7Yv5+7dhIT5e+JKymUohcaE+P
/IGVBsBalXLEgvQil3BmvArWEMpQtyq+/5u5+T8g5vZYjSgvluXN8nCYQ3P9mMvkWpyS
hJALC0e0qZ8YrIAllXR3v6cmJo9s10HaTRekMWZT6o08fLiAtLZ4tJcaMa1GK6zgujpe
iDIaO4pSNjTzpCSDXK+EnSo4p6txZGs8m94UWTHxSIITyxx/9E3K5qvIIQl0o1SiMfuG
CFAMNJaE0mTqOU1ZUcKssIkL+Yu2OQqnE8KVgqvK0RwjlV1+5zyD/4Qf+G7p3l6cWrsK
QUOoJJYXSZFLO/SqHUKakcJez3S6JhC+Eouv13DlpjkUjul7F+WCx+FbuzUh0FKcU1mT
fvsEnl7VHCEpzc8dDVAJBjvikoG9fninc0gJ8/rnr5wZyuwIklPpCCKVSs6IIoayqkne

/cqSUB5r5GoquVzFYvVs3+Wt6qpH16lhsQO4QgtYIi/KLL9xXnlHK7QrwQuWxsfy1Eu5
XOWvzOsYp3blfSSiJgDh/XrmMV4GeLUrAf6E8d06yjEJJKJVyuwLqWJRyT6+lJwYO5Jb
dhQvuK88olcgjUDJOKff/5i2hxyCl6wQklHpB+Oy7dpWmnZKbUBNKYU0mJHC3rRzoQSh
FxqtRhTLUytc5YjJ5r/uqpEUJqCOUcrheAUkod+yQsmkFZFESAaFkxqtACHCoCaVc1iS
gD6EUFqXc8UmALEo58EUo5Zpeqz1KCqXc06JlZPBMYZuNF0kXnD8PHDrEX5NQEn6hhit
C7xal+O0HDvAkgI6QO5EH0JdQKmFRiqzXM2d8O8aBEMoWJSDFKbdudS4uX1cnXQtGFEo
lOjju3K/79/OkPSW9ZzJDQqkXjCqUTU1SYkiwF1haGq+32toqxWu9EYoWpWO9WyWEsks
XyWNx6pT8+wdC36LMzubDrc6eBXbtkpYLazI1Vb5QQHuM5HoFgPHj+bOjUDq6XQ2S7EV
CqReMKpSVlfw5Ojr4DE1/M1/lHkMJaD/V1smT3DIxmeQfhwfwsbTiNyrlfg11i9JsBsa
M4a8dh4koHZ8ElBPKs2elISdKWJRlZVLn02DxSYCEUh+oVaVCCaEUrqbkZHl6h+LiER0
Hbyjpem1u9l6mTCmE27VXL14SUAmUHCLCWOhblID7OKXSGa+AckIpruNOnYKvyuNISgq
PtTMmdSpIKImA2LdPqlIhxEwJHIVSLmtJrvikwB+LUgnXa3y8tlNtKel2FSiZ0FNXJ1V
iCQeh3LKFx9sA5RN5AOWEUgm3q6B9nNJgQ0MAEkp9oNaJ41jZo326dqBoJZStrVL1Gjk
tSpNJmrPxzBn59usrSma8CpQUSuF27dxZGhwfigwdymuinj4N7N7NlxnZ9Sp3xqsjjnF
KxxmSyKIk/EItoYyJkdwqcrlf5XaziYtn/37njML2VFTwtPOoKPl7wVom9CiZ8SpQst5
rKBZDd0dkJDBqFH+9aRMvQCCGPBhRKJV0l48bx5+//56Xs7PbebKTEtarQpBQ6gE1XRF
yxynltigzMrigNzdLNx53iPhkerq8ExsD2gql0V2voTi9liccC6QfPCgJQHKyct8phLK
x0X2x8UBR0vXao4dUlrO4mD8rOV5cAUgo9QAJpUREhNQj9+Z+VSKRR6CVUNpsklvKqK7
XcLEoAeeEHke3q5ICEB/PM8wBea1KpROwhPv1vff4s4HcrgAJpfacPQscOcJfqzHTgt6
FEvAtTqlEIo9AxCjVFsqff+ZFACwWbikrhZJZr6E+NMSRyy8HrFZ+LQkBUNLtCnARFla
lnB0dJWOUgJTQY7PxZxJKwi+WLJFeDxkClJQo+31yC6USLhtfhDIULUrhdu3XT8q8VQI
1LMpwcL1aLMCIEfz12rX8WcmMV4EScUolXa8Ad1M7WtoGmIPSEc2Fsri4GJmZmbBarcj
JycEWx5qAbmhqasL8+fORkZEBi8WCvn37YtWqVSq1VmaOHQP+8hfpvd0OzJwpZXMqgZx
C2doqXaxy3hi1tii1Eko1Ml4BdWKU4WBRAlKcUky5pbRFCSgrlEp1cLp0cfaSTJumvFE
gI5FafvmaNWswZ84cFBcXY9SoUVixYgUKCgqwd+9e9PJgKdx8882oqpCSUkJ+vXrh+r
qarR6y47UMwcPuo5nFHGqtDRlvlNOoayq4u03m+XtIQqh3LePHw931pUSVXkEWgulkok
8AFmUciLilAI1hFL8f3IJZUMDDwEBylmUx44BR49K74VRMGmScvc6GdFUKJcuXYqioiL
MmDEDALBs2TJ88sknWL58OrWL58ORYvXuyy/scff4xNmzbh0KFD6PpLubTevXur2WR56d
ZrEzpMoGcQiniGklJ8maeZmZyt9aFC1wQ+/Rx/pyx0Ha9Ki2UYnhIfT23hERySLDYbFJ
Jw3CxKEeM4EOUWlr4c1SU8t8pt0WpVFUeR7QwCmREM9drc3Mzdu7cifz8fKfl+fn52LZ
tm9tt3n//fQzPPP4bWEqqh3SwPqgt+6nqqh25ISwP69pXem83
AihXKnjhCKKuqgt+XEok8AB+jJtyP7tyvp0/zSZuB0BJKtVyvnTtLHRs5E3qqq/nNLyJ
C2QpTeiI2VjoHW1p4p05pl6LcQmGF6B/f9fOtNJGgYxoJpQ1NTWw2WIajfNSlJSEip
Fr7Qdhw4dwtatW7F79268++vwcvRmJiYtsjXclswkAQIr9
8Oc9+LSpS9vvktCiVvMC8xSmFNZmUxLMO5UYLoWxslFxTSluUZrNUwF5O96uITyYn885
OOHDsmPN4XzXyDJSyKJUUyrQ0YOVKKYyihlEgI5on85jajTMPhmL50a01Tk3STS3nDhx
44oorcPXVV2Pp0qVYvXq1R6ty3cSs3rx5Tk3SS3nCDOieNEMraWu+Vb3x
BKYsS8E0olbAmAW2E8uBB6buFa1RJlBgiEm7xScC7S1EplLIola6UU1TEjYENG9QxCmR
Es25f9+7dYTabXazH6upqFytTEBsbiz59+qCq5FtVuBgyXaqDgd0BDhAD0DRo0CmbQ8htJF12dVrGdcvbgEeZsqYt83odUg9eNFYl1RYEUo0a5r2
8jZeLo0f5RRYTo166dLu3AVit3NrIpgJqIloJpZ4r4ZIr4A2QvnFF/y5d291KpYokdATbhm
vgORSFIXRAeVdikZ0vQrS0gxjRTqimUUZHR2NnJwclJaWOi0vLS1FXl6e221GjRqFkydP4ow4WOTNfLVXE6VLoGI8U5sncu3VJ
xAmdFhhaAAwcOICIiAmkGPPhOmZtqlXMym6WbZLDuV7WEsn2PXS2L8sIF/lCakhJg9mz
+etcuddLmlRDKcLQotXApGtH1anA0db3OnTsXr1+9mWmGhsedxbcW5t8fsoxJJmxP7OZSd8Ce+etWofs8QY98Ku2VnJBbw+T8SJbNpS6GTyJQnw1T5U8KTB7kdOY49PwFgY

Au02nTp3atv5tt92Gbt264Y477sDevXuxefNmPPzww7jzzjsRo9TcfUqi5BAHb8gVp1R
SKPv14xmE5845p5UDyluUCQlSx0Vpq/LYMeDuu6XOAGPKx7gAZQqjh6NFCajvUhRCWVc
njd8MhnDs4PiJphH3wsJC1NbWYtGiRaioqEB2djbWr1+PjF9ugBUVFSgXN0UAnTp1Qml
pKe677z4MGzYM3bp1w80334ynnnpKq58QHKJ0ndpDXOQSSiUvsKgontSyZw+3Kh2tR6U
7GBERPDP09Gn+UDJ2I4ppO6JG2jxZlPKipkuxSxduudps/P8L9nirFaM0MJqnps2ePRu
zhdupHatXr3ZZNnDgQBd3rWExskVps0lDTJS6wAYPloTy17+WlivtegX4zUgIpZJoEeM
CKEZpZCIiuEegupq7X4MVSnK9dojmWa9hjZEtypoaLpYmEx+moQTuEnrOn5farWQHQ62
EnrQ04KWXpPdqpc0rkfUaTlNsaY1ccUo1qvKEAJpblGGNkS1K0Qvt0UO5MXNCKMU0ZIA
Ur4yLU3a2FTUzX8UURBYLd8WqMdZXbovy/HnpWJFFqTxyCaVwu8bH88o8hFv8tig//fR
Tj5+tWLEiqMaEFa2tUsKGkYVSyV6ou8xXx0QeJTOF1RRK4Vno21cdkQTkF0pxw42JkaY
pI5RDLqEkt6tP+C2U11xzDR588EE0O2RbnTx5Etdeey3mzZsna+NCmuPHuesyKkp9l4c
cQqlW2SuzmdckFd+nlhUuhPLMGWW/B5CEMjNT+e8SyJ316ng+GGjmesMit0VJQukVv4V
y8+bN+OCDD3D55Zdjz549+PDDD5GdnY2zZ8/i22+/VaKNoYljQoqBcV9wSgWpWpcUiJbW
IOKUaiTyAuhbl4cP8Wc1YtbAoz56VZ6woJfKoi9xCSfFJr/h9hx4+fDjKyspw6aWXIic
nB7/97W/x4IMP4rPPPPtNfHVU9o1UiD2AcoQRcE3qUHkMp0ML1qua5kJgoDZKXI6GHLBN
1IderqgRkyuzfvx/bt29HWloaIiMj8cMPP6CxsVHutoU2WiXyAJJQnjzcLhL1oJpdq
uVzWFUk3XqxhiAMjjfiWLUl3I9aoqfgvlkiVLkJubi4kTJ2L37t3Yvn17m4X5haahXSXS
MuOFrYVF26iTNuhHohaa1UJLrNXjkHCJCN1x1IderqvgtlH/729/w3nvv4YUXXoDVasX
FF1+Mr7/+GjfccAPGjRunQBNDFGFFaGFRmkzBu1/VujE6DhGx2dTLFBaZm0oL5fnzUuE
GNS1KQN7MV7Io1YUsSlXxewDc999/j+iAiAvuFqKgoPPvss5g5g8tHS9Alwoy8s
DE0rGpJnsle6JZmVxYT99Gvj2Wz45rtms/PeqZVGK8yA+Xtlxoe6QUyjphqsuQihPneI
dSBFv9gfGKEbpI35blO1F0pG8vLygGhM22O1SUooWrldAsiiFNeMPp05JxZiDmabLF2J
i+KzxAPDxx/w5LU35iYHVEkpHt6vawyrkilEyRhal2oj/jrHAXecNDVKOArlevUIl7LS
gspILjdms3Y0lGNer6IV27cqHcCiNcL9+9BF/VsMKF0LZ2CjPDA2e0CKRyCXRXnmjDT
EhG646hAZya8/IHD3q/ACJCTwSleER0gotUDcHNWwjDwh6rMGI5Rq3RSFUIpkMaUTeQA
+fEKgpFWpVSIPIJ9QCmuya1fuASDUIdg4JbldfYaEUgu0jk8CwVmUasejhFDabPxZjeN
mNktiqaRQajmeVq6sV4pPakOwQkkZrz5DQqkFRhdKrSxKgRoWJaBOnDIUXK8Un9QGuYS
SOjgdQkKpBVpaEQIjCeXAgc7v1epgqCGUoeB6pRuuNpDrVTVIKLWALEr/6NTJ+ViFilC
ePSuJlBZCKVfWK1mU2kCuV9UgodQCPVmUJ0/y4Sr+oLZQAs7uV7WKyCstlKLD1tNvN
TCYuysZE/AoUmbNYGcr2qBgml2jCmD4tS3CRtNv+FQIsBw0CSkqU/06lhVJLtyv
AhwWIrOtgEnrE+UAWpbqQ61U1SCjVpqaGly0zmdSbpNcd0dHA8eP8rnE+UAsmlfnWs5qGWRXnsGPD
vf0vv7XZg5kypnJ1SKC2UWibyAPwclCNOSa5XbQhGKBjj16sfkFCqjbg5pqbmSaFCqjbg5pqSoM1jfG4H
EKevrudAD6l1gBw/yC9sRmw348Udlv1ctodTSBR/sEJHWVqm6E1km6hKMUNbXS+52Eso
OIaFUGz24XQWBCKWwJhMSgNhY+dvkjv79XeOSZrM0qbNShLrrFQjeoqyq4ha+2SydT4Q
6CKGsqQk8zyAxkary+AAJpdrowYoQBCOUavZC9KAlSulws9mM7BiBV+uJKHuegWCF0r
hvktODqwwNxE4jnkGZ874ty0l8viFRvXTwhijW5RaXWBFRcCkSdzd2q+f8iIJSELp703
IV/RgUQY7RITik9phsXDPTn09d7+K2q++QPFJvyChVBuyKAMnLU0dgRQoaVHW1Un7NbL
rlSwTbenRQxLKrCzft6OMV78g16vaGN2i1FIo1UZJoRTnQbdufC5qLgRQoaVHW1Un7NbL
og+MXmgtlcXExMjMzYbVakZOTgy1btnhcd+PGjG6PH374QcUB4FexlAKSCi9I4T
y7Fk+YbSc6MHtCpBFaXSCFcpwuI5lQFOhXLNmDebMmYP58+ejrKwKwMY8aMQUFBAcrFpMY
e2L9/PyoqKtoe/fv3VXV6nFQXLmDHeTACSURsCxWo7ccUq9uOCDHR5CFqW2BCqU5Hr1C02
FcunSpSgqKsKMGTMwaNAgLFu2DOnp6Vi+fLnbNWWzXX1Nv9qO/2Xb3Ink5OS2h9lsbLk
wUjKPFpjNPFkCkN/9qoeMV4AsSqNDrldV0Ewom5ubsXPnTuTm5jRk7zF
wUjKPFpjNPFkCkN/9qoeMV4AsSqNDrldV0Ewom5ubsXPnTuTn5zstz8/Px7Zt27zsO2T

IEKSkpGDChAnYsGGD13WbmppQX1/v9NAMvVgRAiGUZ84ATU2+bRNOFiUgWZVyC6VeXK+
OWa/tizr4AlmU2hKIUFJVHr/RTChrampgs9mQlJTktDwpKQmVlZVut0lJScHKlSuxdu1
arFu3DllZWZgwYQI2b97s8XsWL16MxMTEtke6lmXj9BSfBHgMTtT69OVCO3cOaGjgr8P
lAlMqoUdvFuWFC/4XRm9slFzSZJloQyBCWVenfnUtg6P58BCTyeT0njHmskyQlZWFLIc
U6NzcXBw9ehTPPfccxo4d63abefPmYe7cuW3v6+vrtNLZRaWxECk4lblSdOcPdrR0M
vhDUZG6ttpqaaKC2UWp8LnTrxur/Nzdyq9KdKi7BKYmN5hRdCfQIRSnEdd+6sjxCQAdD
MouzevTvMZrOL9VhdXe1iZXpjxIgRY8fuzQG4EIJbld/UYzoYyOjkZOTg5KS0udlpeWlmL
kyJE+76esrAwpRvnD9eZ6BfwTynBMAFBCKEWHqWdPfdTZDDTzleKT2uMolL7GmMPxOg4
STV2vc+fOxZQpUzBs2DDk5uZi5cqVKC8vx6xZswBwt+fxv16iHXL6SZPVuUVDiNqWxubsZDDz2E48e
PIyYmBhdffDE+/PBDXH311Vr9BN9paABOneKvjWpRklDKg946TIHWeyXLRHtiY/mjsZF
blb4IJble/UbzZJ7Zs2dj9uzZbj9bvXq10/tHbj9bvXq10/tHHnkEjzzyiAqtUgDhdu3SxbeTWS1IKL2
jpFBqnfEqIvS2PTowe8vJ08Cft2vD51cPxG8xJ2YYMe45MACWVHkOvVM3TD1Qf+JvS
Q69VvSCjVIhSEMhxvjOHgeiWL0tj4K5TkevUbEkq10NvNUUAWpXfkFkrGQsP16ljdJZw
6TnrEH6Gk/y0gSCjVwggWpbf08gsXJLEgoQyc2lo+Gwmgn3MhkOEhp05JZQ/phqst/gh
lXR2/loHwuo6DhIRSLfRqUYqLrKlJKk/nDlEYwmKRxCMcEL+1oQFobQ1+f+I8SEkBrNb
g9ycHgViUwirp1o2fE4R2+COU4n/r3BmIiVGsSaEGCaVa6NWijI3lZcwA7+5X4XZNTg6
vKixyT7Wlxw5TIIXRKT6pHwIRSvIC+AUJpRqcPw9UVfHXerpBCnyJU4brBRYVJXUk5BB
KvWW8ApJF2dwsuYU7IlzPBz3ij1BSxmtAkFCqgSia0KmTPt2WvghlOCbyCOSMU+otkQf
gXgXhBvbV/UoWpX4IxKIMx+s4CEgo1cDR7apHtyUJpXfkFEo9WpSBFEYni1I/kOtVcUg
o1UCPcSlHSCi9o4RFqbdzwd/MV7Io9YMQysbsbGjucUJddrQJBQqoFeE3kEQihFHNUdJJT
BC6Uex1AKyKI0LvHxfE5RoGOrklyvAUFCqQZ6tSIEYv5PSuZxj1xCWV3NE7tMJkjrcYYv
94W9hdLIo9YPJ5Lv7NZyv4yAgoVQDo1iU5Hp1j1xCKTpMqan6Gj6G3voj0XZ0iJ5H0go9YE
vQskYuV4DhIRSDYwulC0t0gUYjkIpxlLKJZR6c7sC/gllVRW/6UZGGSjdoQlt8EcozZ6g
qT4CQUCpNc7PkptKr67UjoRTWQ2SkdEMNJ+SyKPWY8SrwRyjF+ZySAkTQLUQX+CKUwu3
apYt+qkIZBDrLlebYMd77tlolQdIbol21te7LtAl3TVJSeN4Y5Xa9Gt2ipDiX/vBHKO
/85swvOupjLg56nUMJcATOUwmLujuhgeE+wUmt1Dq2aL0ZXgIJfLoDyGU3jo6FJ8MGBJ
KpdF7fBIAzGbpRunO/RrOiTwAuV7bE+4dJz3ij0UZrtdxEJBQKo0QSj3eHB3xFqckoeT
PwQil3S6dC3p0vfpTGJ0sSv1BrldFIaFUGkfXq54hofSMEMq6OsBmC2wfVVV8KrOICCA
tTb62yYUQytZWoL7e+7p0w9UfvggluV4DhoRSaYzgegV8E8pwcAC9V8E8pwvcAcC9nX1QW2D+F2TUv
jM5LojdhY/gA6dr+SRak/yPWqKCSUSqPnBA5HvHuAlluF9g0dGSiATqftVzxqvA1zglWZT
6QwhlXR0fkuYO+t8ChoRSSWw2PjwECA2LMlyFEgg+TqnnRB6BL5mv585JVjVZlPqhc2e
elAe47+hQVZ6gIKFUkhMneMwnKr/IuNJKG02qeCA3n+DkgQrlEbwLPhiUQqrJC6OF+M
m9EFEhPT/uXO/nj7NY+QAkJysXrtCBBJKJRE3x0bCABJKJRE3x/R0qbenVzwJ5cmTPJ5cmTPGPTZNJvwQQ1kEs
o9ex69aUwumN8Uq/jgsMVb3FK0cHp2pWq8gQACaWSGGVoCOBZKB2r8kRGqtsmPRFOrld
fLEpy3+kPb0JJbteg0Fwoi4uLkZmZCavVipycHGzZssWn7T7T7//HNERkbiV7/6lbINDAa
jDA0BPAtluCfyCIRQnjnj/7Y2G1Bezl/r2aL0RSgp41W/+GJRhvt1HCCaCuWaNWswZ84
czJ8/H2VlZRgzZgwKCgpQLm4qHqirq8PUVMxYcIElVoaIEYZGgJIQnn2rPMs6ZTIwwn
Goqyo4DOwREbqu0fvj0VJQqk/fBFKPZ9/OkZToVy6dCmKiooWY8YMDBo0CMuWLUN6ejq
WL1/udbuZM2fi+fyxKuuH
qD3K9KoZmQtnc3IydO3ciPz/faXl+fj62bvdvmcbvXXnsNP/30ExYsWODT9T9zQ1NaG+vt7

poRpGsigdk3VIKF0JRiiNkMgD+DY8hFyv+oVcr4qhmVDW1NTAZrMhKSnJaXlSUhIqKyv
dbnPw4EE89thjeOuttxDpY8988eLFSExMbHukp6cH3XafsNuluJQRLErAvVCSy4Yjh1D
q/TzwJeuVzgf9Qq5XxdA8mcfULsWcMeayDABsNhtuu+02PPnkkxgwYIDP+583bx7q6ur
aHkePHg26zT7hWNvTKL1vsig9I4frVe9C6WhR2u2unzNGMUo9Q65XxdAsYNK9e3eYzWY
X67G6utrFygSAhoYG7NixA2Vlzbj33nsBAHa7HYwxREZG4t///jfGjx/vsp3FYoHlH
mR3hDuF0vukiftT3dQULpmXBwvQqL0mbj1Xcca9wCXEBFebRwPx/0iCehdOzg0P8WEJp
ZlNHR0cjJyUFpaanT8tLSUowcOdJl/YSEBHz//ffYtWtX22PMHz4cLW
a7htGcbc50l4oGQNERybcL7BwsCitVqBTJ/7anftVxCd79OD1bwl9IYTy1CnnWW5OnaI
OTpBomoI3d+5cTJkyBcOGDUNubi5WrlyJ8vJyzJo1CwB3mx4/fhxvvvPEGIiIikJ2d7bR
9z549YbVaXZbrAiMl8gjaC2VtLR/WAFDZK8dxlHY7d6n7QmsrINz9ehdKgLtfz57lQtm
/v/NnFOfSN1278qQ8xvi1K65n4RXq1g3QwrsWAmgqlIWFhaitrcWiRYtQUVGB7OxsrF+
/Hhm/iEtFRUWHYyp1i5Gq8gjaC6W4MXbvThZE5878mTE+X6N43xHHj/PefVSUMQSme3f
uDXGX+UoZr/omMpJ36E6d4u5XcT2T2zVoNB/UNXv2bMyePdvtZ6tXr/a67cKFC7Fw4UL
5GyUHRqrKI2gvlBSflLBa+ePCBe5+9VUohds1I2pP63oDH
AlWtQQsGiJKF0JpA4pVESeQTehoiQRal/3CX0kFAGDQmlEjBmbItSzBhCQulMMEJplA4
TWZTGxp1Q0tCQoCGhVILaWqleqloFDuRAXGStrTxphYTSmUCE0igZrwJvQkkWpf7xZlH
SdRwwJJRKIKyIlBRjzf0WHS3F3qqryYJoTzi4XsmiNDbkelUEEkolMOLQEIFjnJIsSmf
C2fXa0iLFrsmi1C/kelUEEkolMGIij4CE0jP+CmVLC3DsGH9tlHPBU2H0ykoee4+KktY
h9Ed7oaSqPLJAQqkERkzkEQihrKoioWyPv0J59ChPirJajVOwwVPWq4hPpqQYY5hLuNJ
eKKloiCzQGa8EoeB6PXiQjxkESCgF/gql4xhKN4X+dYmwFtuXQaNi6MagvVCKzm737lS
VJwhIKJXAaHEpR4RQ7trFnzt3BmJitGqNvvBXKI14HgiL0m7nmc8CmrDZGAihrKnh/yG
5XWWBhFIJjGxRiplbvv2WP9MFJhGoUBol4xXgmc8Cfy1o/uVhoYYA+ERsNl4R4cyXmW
BhFJuzpzhUxQBxhRKYVEKa4KEUsKxMLovGG0MpcBd5ivdcI2BxSJ1dE6epIxXmSChlBt
hTXbvDsTFaduWQBBCKShlAgHixJwL5RkUoRoHxzgluV5lgYRSbow8NAQgofRqSoEQJWMf
rGzFGCbgfIkIWpXFwJ5T0vwUFCaXcGHloSlwmUFCaXcGHloCOAqlHSBSQihtNmAhgbv6zY1STcpowmluyE
iZFEaB0ehJNerLJBQyo2RE3kAnuUa6TD7GlmUEjExUop9R+7X8nJudcbGSjcuo9De9dr
QIHUM6Iarf8j1KjsklHJjVHebICLC+cZOF5gzvsYpHc8Do4yhFLQXSnGzjY/nD0LfiOv
XsboWdXCCgoRSboxuUQLO7lcSSmd8FUqjZrwWCnoWSbrbGQAjlDz9QVR6ZIKGUG6Mn8wA
klN7w16I0WsYr4CqUFJ80FkIoxVjo7t35+FgiYEgo5eTcOenmEgoWpdUqjQkloIG4Xo1
G+6xXsiiNhRDKqir+TP9b0JBQyomwJhMT+cOoiOmULlzgz179SYr4SYrzOUXa/ts17
JojQW7ZPHSCiDhoRSToxsRQiOHQM+/VR6b7cDM2dK00WFO+Hkej19GmhtpYLoRqO9UFL
4JGhIKOUkFBJ5Dh50HUxvswE//qhNe/SGL0J5/jyfvxEwZqepa1f+zBj/nVQQ3ViQRSk
7JJRyEgqJPP37u843aDYD/fpp0x694YtQiMgPl4SHSMRFSW5mGtqyKI0GrGx/CEgoQw
aEko5MXpVHgBISwNNWruTiCPDnFSv4csI3oTTyGEqBcL9WV1MyjxGhsdCyQkIpJ6FgUQ
AURG/2W/YwJ+LirRukXkX7wFo9uuMbBUS3KIuL5i5GZmQmr1YqcnBxs2r
0siLd4Y9FacREHoHIfP3uO/7csyd3LGgIRSVjS3KIuLi5GZmQmr1YqcnBxs2bLF47p79+7FzZ
bt27FqFGj0K1bN8TExGIF4/vnnnnPYEwCXzQsSkKiFIPWKT5pLBy
FUkzGTgSMphblmjVrMGfOHBQXF2PUqFH461//ioKCAuzbtw99+vRxWb+pqQk9evTAQw89hOeee87tPmNjY3Hvvfdi8ODBiI2NxdatW3FXXfdhdjYWNx5553KHJcxNNJ+PCfWEDSUDm37GDyaWWdvaVzmMtmVg2CEZmmtrNCxKEXp/qfJIAbHJSnNWRxkiRFqCXWvHsTUVOCrkDZ
LERcXh61bt2LmzJIi4vD3XffrcEvcKC8XDRS6o0ToYejUDLmPgYZSq5XYVGSVWIshFA
mJvI4M3mHgkJTi3Lp0qUoKirCjBkzMGjQICxbtgtgzp6elyvny52/WHDBmCW2+9FRF
69+6N22+/HZMmTfJqhaG49AQoyZwEB3jONXW2bPu1wkF16Qyvp6/kwWpbEQ52BdHRU
NkQHNhLK5uRk7d+5En6+0/L8/Hxs27bNZ27FzD07b9UDUqN
JPIR3YmOlWJ079+u5c3x6I8D57YB3jESr1k1nTXjQSr1e3UMcNJQPjCeWXxAMfTWQZQ+F8jq+O6N53jaWtgH/UAY+WZ2yrUHXT8yZmZGbnsuLWJIiqhFgOgvQYYjnF4hvdeJeFLMsR7rnO7IGAKAsjJdcq2uXZ4y0Q6jGvwKBN8khi/PJf0RhwmNMMYj1fIqKwIUeTv4UfP3uG3RSZhNfB1c3r3bW8L9y91sABhONlngPKJA7WdTvYFgl1cNvF4o42INbQmmm9hB0a5PlGFAkpOq8YlK6iOjjLIzY5yGYKaKhLXvh4eicP7o5THbNqPC2DMmhi+lKhODBm7ouyfQ2GKqxBQNGodwJ3DhPrQypRAOJA/sFmdsGAHHQEJm9N3sGRi9tPLOFuvR7mLlZlhLLvVN7b2x6W3DZEo5EAyTGo7RtKeD8HhRTjb8hKxyCTtXxOOovjeoJIQYNDv29pgpxb3wjB2pKYCU8/fdT8GhY9kBnHY3xFvzZ8XpH6jVMXaS6lt5M9bKqrGm3TbGv+iJXnQFI58qdiI6ZOpfkw7lQc2tRUPyzZnHzYAZq6ZK6aiVy6iXYH4aMIBEcbzGgoQ1Kqt6CxA9Cp9Vg3BbDvPOgx7h8c3QdUrk2e7NMUn4Fc6tM4Zh9V5gBBNn7CKiuSDlF9OmzUn9bTzp8zqWc3+G1I2l6v9lKtm9vdqIeZqt8F0OGo5+A2lPN3VtdeS99EIHHUKqpu6TdG3TXSuAoMn9iX3FxXiYpVcVzoRaHhJXMUtFH9LO0ii0jVnTYzYnbBrZXRh4WHGqk4I2XWU9vz1T8Y8F6tXfPqL3p6jRqoBK7sOBaLOv9ZNrHpQ6fk2RBRzyv6eF0W8eHDTSnAfhz25YSd0EkYehzROsqdbNC2zwWF4IQFtB65rjTDmd4lS2IYkPmrZSGZHHkwsJgvaEfF3Vxmv0xSbuLF6GCJdFe3z5t9M6CLpSIzzmnSf7ZxS77Kj0CHy3X8WrTaf6HbrZRkXX8Lnx7vmsHIENC2D1ZCVjdrjvHzMgDjNjoXo3I3lCiE4ktHOcXm1AOTEPtH1e6gfwFtW85EDRSqvZGHtO3m6LTKP8lz6zLLCf0blmKHMDJWAGyLsc7a6gwQnYzbSzzKm9XZi1sFFJw3hzGpt8k5zlrpLIHLCDmMO3Svk3X0lGz1PXpXSHHx4XNh4R1xyWJ45hX1oG1IMzHO9jbg1pdH5vIaO4s2TqQkqD7ZiLGl4L7z5w7Kcv0fV4cY9V5BUEWd0lrjYY1b3IE2N+gxkHV0CkyPv7FIRRt3fMgMDnB67jNPtkKGcQmuuPvG3uFtdVW6YdyMMnxYWZaIpvXNWuWbL8wL2WI4kmyMUSC+i5s2OMpiN0IgBlNGqtDKQzp5J4Ie0bQUsAFNAwbRnYcf0h4Z7MY8vgO4FLxfnC6DCfIzERw8h3eDAZjyjV8pQpFhnXt4nHWLkVpbUQDBJs3bAc62e5BXd4HKLwjHJdPsNb5ujZ6q7OGJtvJyd5eAzbpYEzymYxCHyGJzHDy1B6s+EF3P+nYoMzM8PuQVRwgqwnNlP0kiGJsfxdXSdIZ4J5cRTHTRxwbE+FS9kVmDYhNhNubcGxbBwIrHbDhrSs+dpmDbMWzzAT+Q0ZUhNvYcWNrgB1a8N9HBTxBy0tR45PbsV/F5qhzHsjsfNhcYhRv7dqWfMzGcAnuXcG3nvzyaKfVJOXEPRlcfOhwH1EbQdKiYGAWyDB+pAqH0X1TKK4ZXJ/Jg4kyYSNtA1Z3tRw9cEVnD7B2Jm5vF1RNnYVfa8nPnLIRR6LbR0bZbvc/7KT+PP3uG9bk2CGsA3qUzH/Ubn5dwbvy9B3UB+7MNGFR5RTRIwoP1xxYh0IEt2cBP7zOzhcpBs3vm9WDCpVzSqqJbrF0LJxjCEB1bdWKEg4w4RfY9H1S7NyHN7cM4hUIeSC3mHWr7pUl1YUB6JRlJ0y8tfcUbUfxoGKLNpO8hX2OA8ap+OJj7UMHtsdCShqThvdzBtr0iIyRaZAMoRAD9Nc4DWS8FZWCMXJVRZKYr1RmNMWjJAXRsYI6S5MkNLsVLn6e5uFiXVbR3MsWjWZlOUMlB/7C/UhPvmnb8jzBYshRbDe0eyb4ZbijR7nZDm8+KGpTTtKGS7UJqzwAHIfwlHwNeHfKpqqbGEUtvOLzGFuJDDAf+7QDBL4zUrH7Ndvsl1FWNyYEmc8Qn3ArOp5SKOO4xnZpxdL6q1wJa3KNdlHuqrMo1XcFQtuXN3nvX/U3RmLPOuwrSFMULhTjw+7EyYOoVlCthM24RzTIENVJGZhPrbRQm1vWj6FS5Cj19+3KZbnjzKVtKsHBdc3Tc6s8y+qEUqwEv1SqpKNmMYb/m7xDsVEDhc4AtfA0W2YSUk3vChv5VIldrHX+F5hxt6BXCw7Odmsd7l9a9E3FNjN6a3EnG4bGrPmScvTEYPy7iVJwqjEefFEaoAr7nOcTD5DMlzc7qpsU0f2UY6F9rUiR9AWPhQxCqO3RyBL6DkvPuKPrDmO75lbq0a9hT9SJHjhZD1fvQxmCnO9NoHu8DFT9GyiwrRcYiGO/iEZPrBa/yZKqnZ5b8U2tNtBSYjD3ix5CBqNSHz5nJ5Kb/4tJAVEvK4X4MK5mL0fDyHF4GaRu4Nsm51TrbVZm3yPXXr+p+Kab7OW1JmL5pHbqpyvwb8zc2TbhmFFB2kymtxV9e8+8ESqAYuamR1Pe/4zJP0Ez2Yq+mJ7m9c+fJuWk4HPWHkhj2J7gwC5Ui9r06GvZ/C8rlKc3mx68FM3cD+5k7iW2WRL3YvtEXHhk6P9kTsJDxP3qbqkmZ1aXNx3O6iOwQW6IgtwyW7hyRtSc9/10VGVaXdPYb+0dE4SUJ2CSuoW93mRdAGJpb6V6oqUrqFT9bNMQv0B2F0Zk8J2yiO2t76S+lJP5zzq4SI6Jb+bR1BlhbOyeSC9NfFxXNqjk6gNAGlDkzc4XLnxhwJeiDeEQ1uB6VFfP6TNCdWNUUqR8W0h0o5vmJx+1Brlfgrx+QEGTQe7vA4/1+RXn4VS4THO9fCHpDkFAS0eThefLRz7VZ1Xq5dLXQsKQWgUtfXq1FjGkwTFDGhpGsYQDlbpOoaLVJYijrZvFv2zRKibg0ntVqLsS8Fuozt4SZk7LmQ2cESc3BVxRKhhyuqi8YbVwKcV1BjCGnOp6hcEkwZWi7QOYvXTwPEd4V8bt3s79/HzeEG+/Te+82wCgyEE5qJGCJ8pTDUg8+5mtGo92yCCfqn1c3Bh6jEz3sncUGBhj9Kn4G2yytLXPOn0q8ZtK1HLf1GgPmT6JwpHyQb+CRlhkB5EaQkUjXodWxtaOD2+9zASVo1yjXVbrGIxqzr0sDxkcJrZKtp9Ho4e6JGW9HsJlRz2Lh+rmDmvnuyzoIbX1nt8O9z5blFwRmPE1hV8+LPqcEfaIwpEfcjHpSJaKCVC1UJXp5dMWZp9HOOdrG2jc6tEJF2JfnO9Pd+Fh2Sep0iSNH0G5oAz/q8qaaD2U+IpaSbSdEjhVoA0ufIFDxSB8lPMrvsDPw/nKjpYQG0ngIjQnT9aYJFc54ssP3Nm+BMNgMZupSSlXbYl7V/o0ZFhiV8lPfMpGCjkb0MjXp69ZZ0e2X7dW3w7LTFDEmzoMOLwcF2HYRj4nRx8AYStvvF5PBe3w1KyYMBgm6yBuRfGKWLKNmtsyjElXD+OK2JZLbKEOWq2gXpcxIuYumsmIxbCebnr7XeB/r1xGjRKnNcCNyaulpyh78T8HqILUQhYmW8iu/7WvTO3YXrdOVBB3fqWvQqtVuK1xdaIKvs2ZPzMWC6bmYXtwJF1JUaQXm2BdLXL93jc4gY4KNG+SLu1lOXA6SbtPECMKUa0uDoWNqg21nL6h8tHV4Ar6p13UjHaOLJs+DJCPpxSDpuRZV6mO2c2zd+0v56BEPmnZuWs7kU2C4dRaRKxcywDXVUvHP5DYJfe4gZKSRwu2zdsAgUu30WdUKFUm9H4HqAu5lt/JvcNUd+q97MWp+jiuHRzWcvPxVjcDyEbmwd3+7tfbh18mf8UUfLMCMwgMEeTH3ZYpSghGCrlwrF1Kzh3EBfO9K9SDeDQkP/LXNzQfyXYkWX6sZcZDlS/uBIXVIk+ghRpDi7NZ3a95KEFLy/6VF6mYzM2hSDeSbhvGYVNkGoZeH/IRrn3c+PHMa6g2i7IkUPnSqPjazAF5bnKxdM7cZ9XhBPcP2f3ghsBL5k2xSBNbVrl/HXBjnqSwsrOMxRE4iUIl5O2hXvoo8u3JBC5tI4Lj3vGUdJNSLXpyWyexZqmMsFDlbqxkGhADr2Ee9lSCVfUSEOFCE2fkqJRPtNdhsUTDpF5E36DOBDk+pYONqcrhMg6aKGsvWzjjuEF9oTBvHtjHWbrRmo88AoTXFb8qiWTu8v1j4Sd1hCGWyqu8AxOrGhXEwJrq1BfIIthYc4mkTgQbgjwfH6DXtk4oVPo2xkWiSrhqXUiTDGx8qFE8jj1q4hXHLbLx9bo+OxEKOCE0L6bQ8+2aYNc8sUJdJhQi8M96Nl7j6mB1Fj7NfBhT5BTNw2GTzuDB0KWBpJdm6IoLLc3jLXQfMyYDhrO73L53WvLrRy4eHl5mCJ8+Ni+ytGxuMbsVY3z8tHE3fh3bYr3u48K17r4v+VhMwgdxq6wWVv1pvNPLTzGGyyVqs2+M8D2DWCBxmcO2KhwIx5ht/MdvP5hf+fqdxNAXcAMbP5F6xL0gAFJzaJ0SLGdhOOPWLDYQ52HFZ9njzMhvqAr7YmZ8Cf4g4BP8i0F47tuD0eBq1SQcyRAkvwG5vjf5C5yNmKXBE2fhEvqMvBrVO2C+WNm8J/ACUSCh1hKhm2SlYq0oVCOIRm+SRBLxvFJ7AyvQhXYlaN2hqQvoDMhk7NR0uE1KFeNmWC5Vv1jNnVxZIB/xJdcyhvhKOyM06HyZuqjg5L1LuUb8BrCGvwDFVQ3ub8ZRc4snqbwsZfg+Fsk6hZ8Ypqu5bkgfM8f0mgXBOp8K4KoIbH/UCmmOXRtHQo2eE7+d3QcOVxZdrwuiTsPEQ2tWLUWHzTzFLvwuVtC23NWm3pyrPkaPMM1jhywdxaOZUJjuHtkMuj2oAlWHavwWPpbAGjWhAbgqR0LHzjSQjI3wWGZsSYK5cxqHBlWK/39hBl2a82tLvJa6LvHwz+LA6EOzjJ/EUmE0otK9jfHI0MLaMJBE0JSAVy5ymVpx5RPzrGqVHe0+wAMyaAZ0HvH+VpwJq9cWbJbDHqMSqbIe6VsHiotNRjNo2dBwvuFw38c/SyHAQNaP5i2UkQZPwhUJ4BBzeSp0d6s2X7vQzfr08JxAKvZyeSm7vXLmoubmrr7QtvqhCEwjPZ6BEO8gQ0RbZUdGBE+wpG3Fvfa3ahW/T2TW4mxshyLsPGR+hcepvE2xSw/wCqp5KwWgvKaifYxY+FKfAn17d02GXA9aEwnqtFRMEkEL+N79V5zDKd62R0kzCLsXFpsfSvXshDQjQcjaWlzu7/BkZJ18+NdVNpQI1yqJ7DqJ/OnnZlSJdKjDtuhVvdnTJqyuMxtqjxzGtYxFL5tf0/R0mLbxj/4gzN91wXSKZafuLqMQVG1vrqt5GkLNiwR/+ebHpAx89mbqHpG1ZYRF8tKlkQEoLn4Sw6LxmNBYAm9Q0kSwKOHD4gtB3jIPQfyPPqsTUDIlZ9aP6WCBdd+VP3LxcmmUIm/RFP84I/wvKjbLrVaWpFxMxQh+BtCuLHC7MyWk0Sd1mnwrwgt0o0SWwQtlPKXJCj1VhRgL4fAp2+tUohA6v5VsQWqsyoRrDYQ4tzWFz8nEHqgSWTe5rDxqQUKJ6G87NfqmV0sPmwJMAUOPFcZjpjP1Q62DKeK01brOLw4Q3PEFbPYjgEu4NnmqVhHdeaC45RT3vkVfAAlJ3jPIFaPBDLHJi1jJkIyxd2vrFsT4rXd6COtnEW13CRW+dwO8Cx7HQ9PyNdP8R6Cz9lwXpGvGDWDM9U4hw5gcKpWNwh/MJJo8cF8WZnHvXtAFVS7V5ZOg4D3Ktg5Tcbyc2eX9flnlSpSHpoZ5mDS5lLgqcpmTeKKWKX3HALoA4EeEO7P3hHHBNn15MYGW0eWQyGxDIfFo9pw+dvACiVoxVwHUtVgbdvTfi4zPv3CxmhSDDhfJ6V6Qi+x9gaR7j0PdPeh95G/dYHMVKQ9FuzLJtdzO5W6rtUyrpjM4LZWEMG8H68dQnlVfo7jMNqkRJFUk5Nb/mDkzSXMYH5tqjCLY82dK6Rxd6SEOFD68lBfzE5oFqBcQUpVS5KrrPU7bRNTrq5O3wEUeM94d6EaZ1ruvxa4BBRmo0D05SYr2q9EL3G7EgLBkzi1MmpZsSOFCZfW4vzcjDIqp6tp1SjDdyTDGh1Ec3hP9NAKJnMjxzP8jxSYZlOkAKfgM5TfRuP9XyO+RGYzoR/xbfP0KnQ7Tnh0jftyPNTeHCfc/yfHsiEqklUbm7L3MQ5T3DW8aAO0ftxzfKw4nRyEtBBkbC3E28/XtdQmcp64m+hGD5Zj2vrXZnoG5wfRQgE9KUUrMkbVT6LjXX6RlxazlOOyMBAXd8UVaODIxAEBxWH8JS1FILUnmm1uB3JtcryO6jAwh6hZIMmOUaDJoLGHu3wAHBiKUU1O7lC2sEh4jj9hGnHKMIMaGJcQO4UPPcWYgaOAdI+L5GSgljuSsmbBpOm9vCSxp0DrA1EuyGQHE7M2i8+f7BZdBGfbbCkm58E1GvUWv1wKd6jZTKKmsr5Gi6VxjJqwXOAx1yNF3Jw2lrfNgEMC4gT9MX8qxJpP8gJgqJ4Gw7i9OV4f/f2RlOpmNW0cbWvTkQSqB6cVTxxcrsghK+wLeu6JYY7uxADC8pDO6pcu2LmnHXiVWhVsgsAnNsy0tCLsYdDw8QiaMFEmS0cgG0yJWBKG5cIq7pAbqxNiRT3lbOETjwNpLBEj/EwH4Y2tbmm3eWeO2W78AeLkQmd5WR33a6X81n5pGLtKTXSmpTgOtsn6pewVvWZKhWB+2KbDBaHqeTgM6Bni8r5vUV+WmnTkLlUFN9IIowOZnNfy5nrEy/Mrqo13aWfDCsgzUTzDcRPsSqjFnWcbXxM7zCkJ1A3ABZgIUNdFXBr7gArcSkQTU+jWADJ96sAJJ+Vm9OPKcH0WFSNmJuNEDsw5T/xpkANRh9pPEjKLVm0B+BwHf6CGUKNGBvMSpvOIYa3D5RVyD56MGw2y2ZKsF2/gxnvUUsd0I0GRXhzhE5DrCd4aKmvMJQDBNDkzo1CHfJpq+6Y72kuZ4YN8fLj6V9aOcI8UGqJnnQk4Ja3mQVxR7qfjd8wz2tBTwJk6mn+3zjtDTXSDLNydS0CMAEB6B6dkg3fg8ZX6cVm4PIfq2nrfMfyrv/L5b7sMmHX4jFHxFqP0Y1YNk0fXJ5y7lF1nS4ukqZ+wL2BwWbJgd7oE3IbA0WADFvXQ9Lb8tjErSr7A2gxCo8KJb3pD9KywHtyLpz9oRAvtlu43uAXr6lUp9bXj/vWGNNwAcsqyHsK7BDqRrK2WSG0I9K+TeqdyRpYe8jvhKeK3MFcJUtUOLbTm8fVgNrxSXb4u5mVFo0LHCZ/7ghMeOUMTCXHOPcgY+PF2r7/G7Bq3T8kIYE9V3kN3+xCEDgzJ+8zl8I7pKOFMSMgRhrIzyzQtq24zMUr2fM/DmvHlJVOMq1SAZpRuMRwrcWcExhYq6WRbm8b9nzoDGnHRfqSuqWoqvNSCiZ2Ns3fKqjYr9NQj5PQj0YOKETOmFkIHFNx7xdVl3AYb8I/TnyqVjDmhR6AOBS8aRaeDYnIOt/xFpXA2FNMG4GdWW0uz5zvQqEKdJ6gzGRjpMp6yZdYtDE7ECCH9XbNvp8ZdAcQb4WZPpSk1vbd1UP3wvCCJMsPZdBJYE7L+QYDKIA/7Fx6ElNg1W0Ad69X3zGc/BZsXfa6FRMLkL4PSfwGdNQcYBaXrX3hK26rhIEDlcGXx5fl4wXuf1i5a36MvTGcrVmjlXsv5/9pnJYQy4Zx8pgNxxnIh4c3YKY1WE6F9oSqWTEDs1c1k8gQ4KvvtM6ZnOaCi+Tj0mzhVxTHrk5ue3Hv8TbJW9cWXtkMo6fWg1LP2ZP51vZ6wdm0DTFfBCW/aGqtSCqRnFjkVEGz/Fn0M4cRfD2AyXGxbfc+uaJHSG0cvgJKZFEHDHmvoe1yTNEeBcTkE2gLqW5NHFYkuG2cAZKcUd0Y7LLt88Z5fSXfhTUyZkaMnVKlSV/2m/NiOFoWhI4CDUk1fWrWOsW/mfzwD3yK9Ntc6Hz7T5Q7PCOG2XOexwgoo+nZqFw6+KJ1GlNxjgd6LfdbfLqIOWrsmBSpQbWfnpwq1x8HmN1D2V8dP68jJoWFTHTAMzEJJS4+Zdju5D3AY6kzBwlKL+TWFsSnSQCMwxKOfV5U+8PTvxLw9bQeAhxTTUctT5uIXBoWyDqsGxQP0cYKBhI4aS5H4OTt1FQ0XOA5o6R6lmQhbT4YUiOWP1Pz4JVyF8mZvmmYMn6HlfhUrFsm3xdljqVNnCJ4wOnXdzXEMU4Z1hGJf4dRUXHiHAAjbuQM3iNsKsQ8GVAAYT4O7aGDrAIdhQW2Hvr5W/g5v2yyY/9Od+Wer+wf4fOUEbzzrz8sdPzn/AI+gmW1IabBfOK4Z1I5mcgbZNv1rqJpFFOjFHqj1rX1NTRgUgxdYMkddOo/Qo2ek5n9v9rbANjbzWxr2cmBeTlPH61ZHRAvH2WBEOzNsNJv7/JBIr4NWBH/cWhO74qcg5OZjNuDlyBqlItqW9uEe3YbVxjjOMKT0K/PI9XjaC7zQ8BLpVpOtb8jgjl/hcp8MvCOglgx5LXzUp/bpNT8RsqrF6WsOy+oS+FnOb9VcN5VjZWXlgDsQvuQF8Xjq0HzAPNzbeqxfaqXvRCvHf1wcgRwlz6J11eBwxsUDDx/OY8tExFNdxvcQV9UeWG1BrQmrHSSQpO2T0qQFpvtmHEkqA9eQZsDRfNNjQpfUyMQv9dN3Ysv7yqVbL/FY6OEyK4pU9JlP3LKg5WmtbwR9GsbvNXqpjjHNlfk2Qi0qBN1ssAt7yoRhYPrjHBk2xeT/6tvJh0JN1xgdw29W8NhdPoGmuhT/rvVQGQvH3VrNJGLW1gQVWgK61SGGC3fZTc2A6+2eLH6FUCgSOVcjYJyFBpgwhcKqjePMCvdLZNb2nRrtuyxAdFm2phHG5QzNZN3rbTNbxkgwwMpnBO6NLCoPZyhpeCWZyyk9dNfOzxCdO1KjN4DjcHBy+IelRySMJE8z1UQUZUtIHnNQ8CbYoA04xKmKjz7OPKfS5NVvNnVgn6Wwyy/+7qS87WOOp4JtwsHpBZhV6JLnB8mm86xEhMyOTgQgNt0NIVP+WDHftuJV1TJpWOw8xMnWsP4SKgqQrYjdSGCqoNthnNkq9YO1Cg7ZGvbt8QRYHx3WTH0gOJjfutsR6lTWG/KMUGkfhoYGSpr0/GeUUKsYnZR75h7GhuCIBEDSDybLSSeELTbDL5sHiUWvEFDyH2cZmbUn9A/ESYnmRjyRTDPWB0xGrUQSxaAtNh5e3pHH9FhsX8/kFSs2P3sP+aDNS2kvcvHiLlcmwd78zHnlyeDj4LSTWJ7FUz/7nmuDLhpnrFOkrCeP1a4n0tHAJnVPCbpt0HtRNKIqg0y0oGlcVJTBn7pPMNRk1avmmp68RQQe8NUgwQWJ3TjXDwLbkVAqkfnyVRJD3zMcqm5oFKMYzHKoJNWr6CO2jr+EwsXrYdaj6jg5LCcLd1aSxqPrY8MYN8RPP0wJYHPP7CmUs9PKYsv0eoGaI0v+GhslS0fIW4I0FxBhHmf4M5Q2q1ydZkO8q1wCUf0b81HcTDXoXcf0VdBQGyVSVJy6gcSbTcsDfSCPj5GLRVZhvUEqSj0lgmmSPOW3XEFNwL2ChIrb2NP4cMVnBZijJ81x0XKcrjWvNmdv8o3G8DyBtCtcEjVxuqT2DbAmmjJSHRSIwW5PFkjJWsRFhgiBDa8QbDXN0TQDSXHsVIYXhcGp3YCnIoPg/+IqbqN2P4l+XQsUn9xdFD0Z4Hk5IZyYcnpl7k4ByNQnfF9E6uKn8rEE8yMt4Rq1GvGAkGLK4Hv8bHrYjQeZI6eDZSeyJYd6wKI5NhLxRkUkTddsS+hfqvBE36CMLjmlCBYvt0rc2fdKJzcoX4DeGxWqRvnaW2xNWcZGk+w4nnaIGqNuRsqOHM6pDpC1tbYJsn7q4GJ6sqwe4L8nVUL2GcEXbL6+5EEOHVp7jnKQ1Rd+6NSAt6EaEzL1HPLVfaa/5ENNaOEXLyG07hAf3OrFHbmHjxDtfUR8Nw3yfDYjMiUYDA7aKgP/kVhbOFRpvNhlR0bPhxEeCs1XXRDDH4vF/tE9dDhwNJnHtahi2CcQU9rhgrw2iGbSKvITYs7/pVFzq7VTc4NWoqi1uRN82pUXuc7kMZrYmbJrYi0FIzdbnBVmQ3pJ/SHR0V/5UnE4Q3FMpe1XnKPXFd6f8CyCqBsz4Awhd1sMhWw9GBJjYR8IN4SHDdJP1hIKkMgDVPvYdGG6IOj6R5bwJ+UUnhJu1TeQJC2HNxiOg1YVSxKJgnvLpK4MBsnjZ1P0tuZiFmMHXeYWqiuvp+rZnUNjlMSl0GbcPXE+cyXN+/IpGlR52MW5Nj+ZdSeDPhoQDFVLu7s8pMEbmAr8yIF/GwaNvoZ2GsqNk7KjSzgJaNvZD42i1B8MQWEFu8OR2cXtEsSrufRwZ9vZX5x5v5qZ6uy4Fa9FixTSp2u/7J71c89NqU8ZRYCpzqhE3ScFrgKzMxJjhNbzC8tPzU6g+y3uFnc+7HPVWXXwLSTRcbWqWwxsCKw1+LANQzdEm6iSxn1htnDbBGhU1CkoNj7ugcrF62RlcZTrBFEnMe9u2ZEIjMcFl8AcSCaZfx0uCCn0PUr3yoh2dfSv6s6lNOXlCafyxWWZ7VZ/Bg5tzbzIN3ORD87SvXZsKjWZ6uAIvTeS0mUSJ3ExbWhyiYmkT6iyh/IAYOZi+VMefbmQ2PzKtYyHsZ2ohqmf1DD1L5K9lnCqnXxjpIyzkzIJhV07AfZwjLNB8N/YQz6p3WVoKx3e6LVaeXoj7G6pjgpNUNBVjUQ2g1Ay2WYgWGu8LqqmJXZEVUTn0BGSXYyr3gLNcJwK1ldd5UOzBwxn+Q1M7mFXnAlJ4N0LlUK/vJZQsPX0e0ES2rTSS0Ck2FMFLHRWrEZeTI+QdHm8TvcjwT6M2RvfWPtKHvNc8x5F1GJmVGgpBy3wjsVXx5Yw/Opt6VwjjxRUFzzC+RWg3x1ylZGzLAnUIXTbb8vP66A2/PskQEeOE1e2R7UhbN1bQIbfeahdQ3J57llHlKjEvh44EX76pHPyaG7DJrhMhGCumR8+xZQDGSEC42YUhcaCEYh46CV0zlgnPLHzrUSohUSpgnn0LkJhhE0ZEAdrOFdWRAAVEfZ3M0BhEvmDsfzZSY6BVfn+CXmTJXtmpkrGJ+hoT61CN8ZOgjHVeR4Xq13qpMWjnwVzx4GdhL8WPpQ2tlOPEXJVpX7hoO0UnlCG1JWH9Jv6z/Lm6Z5UWVoTwsHHqrfuh+TdkXbKgz55pRObo1Y6bqlCwdCJmqEKcUj46g7PLG9dkJBD+rhqU+7iJK6N7HUDYCunpl4+k0ZnJyzGZe14lq1Uk3SoRdfzc0uL0LlplYN2zwsY3YjRAUMHgWLV1/M0NXHf0PVAIobAr0J5TQ7JOh4i8cG3ASnsIxhLe1cdzqQTpl96P4G7Jm4+Txpfrsw1ouHxmlZOrsdfTjcRIcIPWsnsPcdxS9AjTx1wbZT3fOFtuSs0sStTOBuaeIS2Q8LtXLsE0TthhF0IS+MlhMAQyGXIYLoWjpLRNXm3b6xCdG/ZPXBMDBfDTbJUr4FlqPnWNgvEjbuHkDukZ5iR+gg2tWLAsjvuU+kq0wdR9hpIXWHU5O7H0b8sSw4uGT9vWZymSXe2SbmSpRZrlAmLb6eXa/H5Hr9vpQYHZ3+oJ+ShlP+eDGKdz1A8RHjpINMDaOWFk0a0VbqJvGNOeVdXM2LLeiPPfgNgKUEuwzuf2P/AJcIGj4yHEP7JBR4gmz15U2jGd7yWlSZ6jqeeTAOezmZ72s/fE8FdNMyNo4eVpSOdz6L4vM7TGVEJBNz/dSumBYx7Zpj6EAeDCN2fWzMaNTgjrzqVrjk4o9WgfebDQN2m6Egl3GK9MNbUN03r2sFNKaoOmj5G9v8Ozog9AD7I+1ihJjzXAntb+fYa8PYcJs7RrHJqlCOwB3u9l9bYNzDkWdlIB89UBDSiMwI8cI0jtz7Lg8KqkIdlTiAkDC0JnGp4JhTg2vOlLfKmdhbKhV6sCOewRwbZHgAPsw/tlMjxhGIA13i6wEHZQwIhPxrmOA1FTmNsWKpLmYU67MUf1DuLBN5HlD+CWZXuy0T+PYqmhl4GuYrIzYQb3G5B+DPSxzxcPPTDbZUhaBzfqDDaewqN2MpLBuPoAhgYsNkS6PXqZXrt3qqHWjt0HCIm95IsjX6PA69tRYmmQYpE7T8XpQWQDfpoabIeNw7FeD6PEvGjLE6qmIzmGY6SzYjGQ8HIsrxVTHQBLxLSlgqtK0rhAz9ceIQkq9cB58qBRoudoKEwyzLxRaT1uUlIJxzOGgbp19gkMv6XNLX1vmBGovP8j6a9+Vt7bS4VUPSbqMvF+UoqNXKwS7vqw9j9dUGHG3Es5oaXPSK31+iMGm+Gd4eE9r3pvbxQJpUSgSB5BBx5EYLNoqqfYMH3zT+8Y1wFfsHHFMcdrjdcR4xl31RJm1SGzAtGh3DG2z8EPTlNFEfzTeJsI8BrhUF2x+o/cH80DD5/l5fjkQwqEbyv96WGwF06ezSDCbAw8xbHrGuoIfAvuRcSHRPIEcEuGUowPvMhRntLJFJCfJ+nmTjM/xnlFEjNLPbgYh6hfbNvjUwB3VyAwqYK0NKtl2M59ElXE2Vzl6TpEWZe4XhCfrNNG5JVFZdYkmAl9hYdLA4D7bIfONvtMG3xnkDGHStfybKsbT6PSwpvUSAEG2r68dWPtrIEo8yXNYIaZR27zyOF5tVjQgtv3LNw38aVoNWhExLHxugd4+v7wNZJ6mVBbTnwIc9iLDLzQeY3cTqqcyKH0PxTjpbpaR4yYCe0JWZVOFHtA7FCu2qXlrbeHBTU5kRdwyvjXVsqeBXjBpPFjtHfQy4DpDgGhR9cmlD7Yh/mPG0CwEmbJDyzeQATI0x9MbhjvbQZ5LKYdzPAk1EEq/n94xwjY9oiXxNAkyj7ixnIOxJpfSSWNEP0i+ToTSJFQqU8u2AWNgHHC0ahshbOCUwU46r3UTaAHVOtkFkgNT5tUrGlJH8q5hBhvQ/5EkOTIr8A95ePYNaI5e/w6fjk0pQsfHyHd5rqZe+KCgZBQ0rSxM9OjcuOW3mn2vpEZhiGtCPjS/+xTEjaQgnLtwHwMuGs14HhYwRl3BYH3FeJU/dDvHhxyZxgTH9nFtGwwQdY43G/SAqbk6LRtBxhMKp/NX1OjoPsJzYwwnOLRV8tGuF8NIe0iBdh9xC3a7PdcKRr8wbmfQrnnqLvU4vgUr4GrSZ6Sx9zV9XSRnxMqNiW3SrpLt4b+wW8h+6qiSmFcxKKlYINFAf6R93uh1FBvVj4wz6YnAzMAFqqzeftE63eNK4XBk7xSk9xqDFIOyGWENaVaTQakHyq4VeRrvtBDSpJWV5AgBUybBXxkmgBvbXaNDtnhqYcmp6ejsGHBwjG6P0yn3jSWVGqWt6mWm2H2FCj0aAYdX1PY7tBijrfmHbsX0uuLA1aQT6kbnP1S6NBSqBfmWhfdgPjEZzFlnpJRA7OaeBXQtv0yz4FqbNMKXBtgVtfQ5kY6zVDkpq4X2Lo+6u1Gm7QrrPXG12CcwzcaGPjNRSHhdgOWp9dFiQhM5yUfNwqqyMVZrv8N3E+PA/hhTLWGnoV/jmLqjE3wVCXtq0vx0VJAbMjqNPlY1E8EWmP+AXQPGD5l+N3CrfhfRH93YXqjKoKMwZG6ZNpR9dUvYXIAHb5MYcuzzQfzlBu5DAxDBUlkuD0jGAW86J95MgFEz7UgmzG0uA0qWP8YWmbc/7+PUpHDm/SkCG2iFU8Q2SVOx2qdBU3m5WqgLENS5eSQYdVXgVO6l5xPaXE7jfgxS7/7XMAPy+Dme1WJUhrRqhUNlOUGHGQ4cLswUdR5KzgCRbtKprZAP7NvjGvqvJ8lBe0KCaNXMOGc5d1g5FIfg4bXlNH0tnPOvlfOf3E+JGhwqZlfeU/PnjpxjMROwBUi7uB0xPaTzaNTfcP0TG9MOAYqDUVvBu1eO2LsM3bcMKJdGaAKfFyYAAkLW5IvsiIOaP0YEqCEpnsjFc9+DMXE8bAqMIWRyd6R1dWP+Xuu1FuBH8M2wqWcwdFJ9Je8YTBfZkYP1JNvF3JaEfTrGgQK1zPwFGtF03p+wIWYvP2fbKU+1gvW8+2gVbIYvqCpc2B0T/KjO9r7bcVA+pr4AeDe98C7LqSZPIsa3PNQmWJdBvdpgnHt5GnaQ6SdkSs3cVj68GvRUSYGxdIkgaXwIfUYLBkT3Oe5gBhFFJqJeTr3DExk9IgNPyMMxQ8hB7O0RiMZG66+GZFoHzO+UZ3+BQ+K+E7mhnyTUhjOFtrs8LQmTbFSYAeRVIfUfqcqvFlwjr8ajHY68STTFh5v92x2wcW4VmSl4Stn5D0Fb3owKHJ5LeJS/r1VUG3YiABiu9WdJyJnGtd05sQBu5dm9tsfJi7P8ic6fFsAsDVWhAOcrlhHqhr+UUgGSc8Q8GRnkvrBQ1+ZAJeRlJHepLuSm8O2yFS0fI0AdmhIaO9Hdsd4dhXa7QwxM2MmVyJp5K0NOfv3iyT08d7nqsIG6V+uz8vDuWqiHtj/uxa+V/xWBXeXkxPH8RZGfyQ1BGIhtzVOxdXo73Zfqd3TqB0CB9Lac38xwHdyXaEuHbYlVgtRifXbpYq7GR9MZp+gXqTzmLMaQDwZUx9LrnDszTbJRc6bqCBGfpxuubyAj6VIWwW7OH7CnlGNhVgX8C6nWIeEcQ4hJkLRV1V5Sy1Xe7hjIpK8lCdj3KKDNNsglEkmmYuH2C4k6FczvdC6fx9ZwYpjYCl8dpy8Cam8lRtcDtbzTwp/mzYWa2s6tmzoi0+b3Yl9+RTBBXXZHDrtUPqYyP+P8cpsTwNJJ9Vd8Rr0SqOxiJTl+O8vuNOCvIDCtg0j7OBZ7zdKZEiYZE0wAz3AZbcAcdDOPM0s4hUMH0nfNP0u8s4EhJjR2IjDNg5v7f3qqaLJjqZZK8NVQdS4gJuYAB4hFsSPaVbU5vUBkvB9uYGflQmQ3HCzVjJb0pqTXeCqNqUcx4SZW2bgjHbpWSNJqNMvyIqnbSm6s9KZbcTQqoMTOXaF2k4dRxpIIHuWWP6ybrSlh6S5LBbUPg+rIqQf/KWR0EnIqn06ZwLkFZMZLIJcfZZFuVzrGFiVZeJcwTmP7XozBX/DGZxj2HrT69oMWbp2P8R1sPcJ29oNnHzIhn1mPl/sxy1PT0UIkg25GQ9x2I3Ffb3JH7NEu7NK0S1Iw6H4ibSCDp1P40ogmA5EEc0I/UftEm5vyNL5pkbN8oZLn8mzEhEzkb5+E7J3OAV5Ol13rW0EYDg7hJFyMvvgJUJ6wFh6D1n+aHnmdJG1CY0g+ivS8aShpkwVPkjjmpgHohy0H5vW9FnGf+HtOkrsXPpr9lIWqVmTCf/BUVtAuu4T8+HL1iqKJbKQFdePT8JqYZsGPRXUJnx6XQRsdHPrf4FuVB8OxbA1ij33eowm5PnfHnS0sSs+GqlCEYyTiVgpJzz8j4J6qjVkKuGbRZmxGaxsv2RYAbjBdbwJKH9QI6DAbrs4VQb4Ro4CT47YEaD5RgKN26o1R+2aBSnJ7YPT9XwQ88mJQj4G1Aot05Bh1RTlM6VY7t0ubT6f7PdWbfIJBwyXf04S6d6m4HDtPEMSI5ARPSSjpbvHS0yTZQH2uXTIAiaEKaXr0bQcAI3tHZW6wrVOSu+Y7qAwbYDP+GV1WbrjUrEDfEzzFN/Snn9wAvFXJ0c6uxGLq0Y+gPpXzzHS+gJ6HRmb7ubYIfYCZESPxRxkZGbOGz3gTLDUS00NQiLf+wNB6tDuWnDBNNUPD2thxd8BntlmBGG0S/DlqRRHtUJBI5oZ80o6SVcFgb56ZBHOJMW8x1h0OwPkSyoMPBhfDXKtvNCBcCKLuW5n8x+lMNYjEmBnfvgROTUg5T/ApK2wTOMaJm6ScBU7r0tGfiNu54IY4pVnSsHCjupo4uFSbtKeE5AZXmtXDxYsWGAKIAqvZhwhW1aSsANbbTG/n0rCu5rxcFCtylPzz7u5bpjMZcfDb3bJYGpN8n2GkJVvqnlNs2PBtoRsywj+CvpNVOZewQmRuImh7kjYBiRpx3WP+5LrV6pU5yjxQWcbMuPUTUbmPWc8p+H7iThpkfnq9p/x1i0jRZhR2H4YVe7Z6ddGkj0AuO0YoYvbLZOTxrhrUBI5SKuSTfJgPaN5YoFlSt8TKswuYBq0S+Ay2XHOm5l8qH7c2zHfY3RD7Nj9TpEo9qwNrIn2zbY3pnTSKqvDB5vg+eqI0Pvw8Ue2vFtQ+8mIr0rHHx3KJ3sQmBQO/BKx8oMdp6UXzm2QAOdRdpkuVnH+OwJnjuNVaDLzHEKyFyrHPGLgI/5ymZtuVDOfsFdxqFbqSW3uMl+HJPo0D9c7o99XHzVwRu9ZGgjoqh1uR9LUGqbr2JXSt4pBUR0mOcaavY0Kl06WWsHvHsDK7wFVpOmufpJ5q2IkvmWyD4vYVgbiO8MB5zGVgvv4mXwHOcGSZlT3MtEUGLyBhrAbIXBwQKKgZ1z0oiLqdgw8oqpOqbbFsvBQfHX99jVg8QR8EBSDM4wOvXqL9hCiEJPXBAhvPmwjxV7rn5AOEtTgmWpUVl+W5y0Xxt1ISo6cIHOvuWH8P8WLl/eyY0xL+2y1utO5ed/wgkpW3rc5KR7Ey6qZYUCWAWHzPw7I8FYrJkT4ADeqMCI8bTJe5eO0ab3ahRa8bBoGXpQIHHHpL4aTeqOuOY+CBWK03Ujz9Bn9yr8pDd9ZKj7O2SGtENNE87bVD3A2ZKrnkMn8HLHbQPCqgMk0jNG4dMIuVbqLFdSEdr8BLnRdBiiyV43fWMyyTNUU8mmcm6D6JO3nR1xNh9uNgXuP2OtWxSb1WUTSNyNqkrH5BEuMcVzR6qoPVmC1GlhHTDtYIgZyuuK7lPG8j+qm2b7tHbLbCcFW1kYfadXHhCzdWwaTxqjSBY7BChFBxtmD7cqSPyj37rLx4JwjWXNrYmrIj/2KiFKIBHZ4TtUZ9QyUtxOs7FjXYvKFYwnhxDyGXKKo3DUNUzrFuT05BKCo2BMvIAf42S7qBZDYlZhW5AMAzVU4PojDxhAmo40v0eEVOcI4F7Q5RjsvkfuiWspxL7qJ3c1BNOvCUP9Nr+ncJZ0R4tojFsh2pn8VGBqO0zQyXhJxnsjyXGqAunZE+zUYdK2Ke4MOpLRjpWFvCeY5z7sc+3OPYrkHB2U0V/vxtiGFwDDt6vY1ZZrTt5o43RgMeyBmN3V+7E9r/AYVXE4nxcLLMomPa+7CCSXBOQy4g0iKwshqQ0uCoESLdeh4dmk6jgypBePNjz5Fxuc5O4zzfOAkK6xSprA8qmaofHmZgA7YFY9uKTomOfe+bJb0Csq3n3xTaH1qNM2RNlj7oNfLA7DRvMvKIw5Q0qyQtWdP6jgoCm0GKpm/PIKR4fdeGQ5cxWNFG9FxrekTnqMXwc7L73bbMowP7JmPGaT8Unlms6V3YuoWFIQFDhjvMsxDU4NaXWtK1+YhqkJPeaDHCzTp7HbDc60j55RdULJSWWMyPhR9UsG+ECzFwWL9qYhU76IecBHUt8wCohR5jzrVWmqlj4VJPQcQi7BqW2OVuhOOWLIJkYI1VAEmCchFYHfsyT56M+qjxeV3P1nXgbl7ZoudN5PQeD6qCrMzoL+ci3W/CnYSMtJjEo93ZctrLsDKTpWhjohEdVBaWzg9amaDqTlsxk8+Izxzr2a5snl3GvbIBe5wYaVpUTyDrDZy01LlKGmIWWHsHE/0LnQvyWQFIfjp07BLVefMlmmFwhvPZ0FPIcd2dZ7AmFeAkhKZpwdu/s0Cu9Axb+aL93Z6fW6DvVimgJwLW2N4BwBCWvxrtVj+lAiVmRuZuqHgXpbRbKPJE6Kd6KlTWwxTRUEWkHJkF5mAm/VeQeRGyd8bfkhTAc3zcbPMX8YH1gKXgNqdrz7hsoy/17aH0uY4PiAvCfjspL8Ym95Wm4EyuQM9wGL21tRmI6xtQgE09X8lPnkKNfmJ9F6HxCyRxE7lPqCj1OvLzR5QWB3EGA7hBFjgMkUHh3oCf2N9jV8IcA9IGLNVRw3U2hcBBbFxwrPswkx2NIWTzAV6wbSCs14LDGBiLHo9Vf9lvQ4nWqugLX1ewDQNxDpjeXIVFSo6lO+8yhRFcdjKNrIfQIH8PXO81W1jhvMHRdt0uvKL0afzwBiGdO3qWYy6FuWsn4THnsvUHbSJq2gmmPpEKHkUwJCNSrqe09krwHfyGzfSoV3OyM7z/WfKzsUiBdmqDF3EQOlOUsbcy03N8Wy1dI9A5u5VFvK9WjB5h1hX1ozGMKjXPXD9QnfzpNFxFHJoLhk7tWTgpQDaVbhbm6Nk7gKQp3MmCTRXf3fNOpOHfNKyOk4gDqc01kxRNRU3PuQJRtkRZvYYYmYhRvc8MDQDS7JiVHpXt5sNTqKTE5lApQEQXRHqN3hGNpMX2lpKZlq1n7yyDV/RbKJtYsz9gU1JSXnKpFeFFsOQdAR7VhHhhw8Ss5RndvKV2Mor9MNWq1hkSoJknSyp8PflotuSBPpuu9kPv4+WeSgc3fzq0g9r+9Zz7swf8LwDSTzJ9ixdGGiPGHkpwAT8jjHrZmYz4cG+yBSE2N4y5N6rNZzPfM3QBu6VfsyvyAGjYV+WnXXaJqCtUcjSBAaJGeR6Hcu3BDbPd+n5kCyNzWuQnVC8m5Z/BOsvyxfaNR4NrZhH7mG/uFgBmxwf2XFNDZ5Nk5mGr0LOdmO6N6ELvXf+mCCm9S2xmpv9dD1a2MmuurAKN8IRHDNmbbtdt0y1DhXMZ1XLQwXnfXapKGVWVyJm3Wj3hF2eDWgR4J6sN2KMX4lrzsKkw+bm2AkKkdBhH8IdcDpy0Ga7Vm8UNbm8t1Zc0dK6Nd7POkgaUxsa+i1b2V5UG1lQsUuc3N3Mqo1L1KFhsUHr+J3YIVuCuLhRrKQnoj8ZBj8eu0+YDhVdJQJ2eVZhMdxS89yqWEJ/bV3ivSgN6sa5edjxeKw3kQmGd7QdbNYDRA7eHujK8YoxF4YXj+jWsTD9aYWKjrJcDp3LSBnCEDSDaxjRHMb/VPaEIM9PHY2NuO3kDBpEVK7h46hiTiWwVsKXPSJXpl/uOsPRYTp9vZJXOH9X8WBYz/SMFq4sRGKOYnHZWQ/wZuwq86TRZOZB6D25dM5DBCFVY91CqXMTkq0QMc/ZZnuFo21k0CSwiBDIWiA2ohX93eEjbKhFJSFgNsz7q6+N8bhUgs1CLWjE5cuqGc1Xn/91I2G6ZPBMx3gV9qYGTN1nCVGZIrjKwgcMCqoqsw30ASuPQYe+1oSZfIZo0CAgL2U4Aky9WWNu3ml70lHS4f2Fub9kuTBRQubU90+sXDZ7e0O/QVtBRdOQcNaXCQVH9yOgDDvN2bVqhkpIGpIfzl1cKwh08uqRpB6Ye8W0f/J9+1Jd1evh/DRRDhXyhyPfafd0Ry6SZaGjhmf4MP8d9L98jkDWfh4jIRbpwn5NmwAuszJZJ1GC+3v/jMC0s5pgGIgPphw2oJRGdSnywmdeZ+4/8bWKy1gW9MpxmnOhyz/YZIzH/3XwGHvuG2vhZwMAAsA5rDiJp9Thqm0NuJxkrLYG3pDEfAqdn5HvkbVWUgxn6GRfgxlXgAtOIh3zn6EtMXzMlvGnDb/6Yc5xFaa6VdAeTLzY4j9OY5r8C0gaYRFH8IVsQFHbPi4o0k9dYTr5tpXb8S5J0eLMJm+3vHAqDqrlU/fS2rtQZ7Xdz5Ln0IbUfu4xC+jWnbFqw48vUPoyIrZSiQ2Ytbu8qmpG1WLADYcWWMg+EaEZfJAyHgwPf9E6R0ILgnx7cZjLdOUtzXhxbgrAjIr4Kc9BmBJhxa19QXpahGT8fC2FXGwbRrTIOC6AHvJ0AQzgDcZ3NDTYVvXlqArQhFuCSWI8O2IrcCNADbN1ZqkaKkRW4YBmtzbpf3LTa6AmBl9dKVAPswEk6JjhgE+kV7C7dmujG6Wlhn6HPVVIvNfI6rNtT3G5H0bVPaEEIAaD4vRjmhQ8BNl+3yzg2aApdfVKo4jdvHWCU3qimxQVZoSyXOuvzU8HfX8GbLqJlH8LPH6H3NE/Vkzr0k9sAIeRrJiLgJ91jMtHdhZ6KNyqTtUTzxvoQOhDD/Xk3LPBprzwQIBAb/HMBaqAKgHYXvVs5xhbqKGSR9rhUngPVtEjlt8rZKAIUvuqPCEpDBYBrpUM1CWbV1fwX1lCKqOoGzLs7Ms98VAbJjhWJdaq1xVh8LL6LjnDrcO0w4qVFMkRnwmTVHXlQ4gZZR4cqaQ+bw7cIIbjsyJz9THuFmkk+ZEsoxxHxhThVXXEtSrKMi7bzIyD9/XfIYDOaN0wTU9a7gPFmW6V6ZHXHiJ5SJB/AcmAsu+MwYPdQ8q0pUL2ZyaNq6XFG0E46dN2+DmAfkfNsuPM5J4lqD5vGSbHcRWZMhELVrSzvuNCutUgHQhMjixTvxLBxNhK1BjxFQ8Wqy24jmNL3N4bMgK3dXdJDAEHdGTQjUI5iuCSjZ23cKI9Ta5uMAQFwyUOyPHrdMKbY0Y2gX07mZTmbM8KsV2u+nn9nv8UaR67+pRmHQsHT5/99v0Wm9f3L6tXMV0x0QNhYAl99hXc9P8sHJDu6yFoIEtUlPL6Cm3j3nGf0YkzE1Qjp7oAObcISIr2hNMEpSMnQITzgxCfu7YbA1pBtmlwPDyTxBgKQDp7BHGD/+pDUKLMGUbUFT07rqX+k2YQsV7iyXHXJUt/N45+E1C2xOMQGbPvS9BIXI3aDl0TdeXHk3DWWpQBObnj4DbvZWMDkYZDoORDiSuqVh5o7HW99dDHR+SfSA1x2qIvQjORYGTqZ1uDGuxgJ4h7E/EA7tpFvuiSXbDlhRXWzJu8IGGPmzAWb7Jl+FtH3FNF5cXqCsQ/+L4R6r4wFxiYtfFVT2fWy7K4HB3mA4tpA8I7RqEQ0kv+wNmAzZsQPo16dHOHMOLhRGPr3xzpP3FhMd9s9C2Zx0qqEGFvHwmW3vPpGSQSqqChqEdm9BqArgZzSFPNkpEbdtE/KRAQmlv9f7QJh0ipCgOe+IByeJTawZQZkVT2rRvoLN2nFB+tDtJJMM36UjmxfSpAzgr7sDGjRTIyyjmL3jJyYUQiQY/4WR25SDmqL8wlh5LUzaT7OaI0W+JZ2HiGHsW5nlLfGoTE7QbTqm9hqgqYYPpldcrx7jrUTxn6fXBBk+kMgEScb+XwRTNI6+7HZeSfR3LOo8XZOQCUhj5uHLyC0V+T0iaCdD/Ot2ffYXl5WcO5tNtiaK4SjpNJlNPRc8Wl5kpuNGG1D9yG0L8gN3LmsyPRhxFIWhnWqrwrPIbpKYUJGjYlK1EETJNk2+W7zyE52lQChhxmhMjG0Ddd2iAf3lMQkMp5ETeobHlyZGZSwHPaDvzQ2r9wJiDUHBHGVU3E7uhfdAdKPnLuDF+k4jnnhFhFsdiWrvTAeeYWUwZjMsnX1XTcZsHWo+ADPrxsFWeMq4rGMaugK5oRD1w3wDyrX1aEcJmnDX3CKaeaxJ3XE9v3uefJ4b7jMwbk2xumB4DQvLeqa/MOlZyKyYVjnhB04n2WuP5rtlKy4sDeSxOMiVpHyLuhmX7X0cKMFBUUZ94ZpjRMUM9UdwXBYjzKrB3zOqD6ijBmjbEH19D2LhfQ4RcvmErPZW5P99dJMzFSHBqXzEdQdmmEEF8NaWM42Go3VkSY0MaypRb8QgpIqxphuZyJMfHvFR3TfqhcmmiWNw+tB2nQwpA4OaIt7BQ/Rz0MeXZVHgd5mFoxnSEXZDJ8pYpqd3QjeXpt5h7o2M/cWUgGxcnSKeUFAcVyRT4tAhQbC40WGPvh4b7AFjHiyqp73UNXvtwuODl02SQSIRYK+UUnVmB5uGjFwm4YExeCFmN2/U2bUCbR/DZgyBMVUm6FdSqpOrfzWmBYKYCOZcGYhdLCovS3tLbMLQHbUbC55ozlVM13RaUDiCWgTeHo1lhw8U6LKXQKjWEbrZ7kKSLHeEWfyKlXVCAvpcutYmSrPU1cmz0FW6wuzrnLQVeLEu9F7bwtwyAjAS6O3Tc5C7TGQfVsGJ7w84Au2i81CjfZJPd5xtHG9hgGhfXKwdomt4x5qLVQhqq7zuhSBfMiGkdVpnLMqsaQX7h2WJxsdAgt3Pm97MNRJhXFi6WBqJfDS9/sAKpGV1NFuuM8G0JKLMLhKvlxY6O7HQ4dgvOEjUjKVi+CV2omJSk8JfAfmnBqtrdcShgAQWzMGBEdpZy4GDfU0jrdBT8OH95o9xAz4RbwS2gPLnBn6wCvwRmJhzBRe1aoXC+yP8tqiRlTqZHzzwzMDiehACklyJxHXMuj5Zq6AkVi8Z89jSXXL+/uqgF8lTqzw3x5g3RX/xGWOwr8kTCeXRxsWdFk4Hu6K5pQeD9IY3RTxpGWZvdNAOvjgFVdm9a7q70qUP5XMStIfjkGxXd4HtUhshI5+eJL0ZOJi6QBnR5o11LqLy9W8uRa9mm4Lbm2Pdd/KbbUJ51GKtvHGI7zQKmbrTTAxx2V1twTIWc+GXhnufZq3FKZ7jDdlOO6ejU/uVIrXvxrOhyy87GJK5hAHoAkNrf0hNmj62a/CrGL0QFdfKJxL31NlcV7NnOPgnPHwO8P6DGw+JH4NwAw1LRFeLZPtxzMtqHtxmMGaO9gXGMdC4tYtCqmiXTYTQ4z/F8hg59jQMZvpjPBkEgwVgIghlSXsY8nCL+MIbcAdx37V3qlyRd5QkeGnkuexPWrXeuHpO/4k8HkdBWSm5bDVkxjkmjrVXWbMI60lSebpBQcMAJ3tRr4vCohzT7U+vJbSvxqEQnTQlSdkN7L1VWapTmGXUYRlnxI1Q7gfBdOiEPE4cpi38aLQ0C9r97Vf8eJlwSJDZwzLtv8UcN2gpYwEr/2Nx0G1IVJ68d8hwSnnMK6TZsPKJxj4Q2s3oE5LYIhjZs6p0n3STvKQQX8Y6WvdaUUzrpqBWFaTCzoDa2l2P2yRwHhxgJiG1kSFWUX+Vw2bMEuR1UuJ3zbF3CFtjm3zGZHBTu/hdaE6fxKHLkc/N7UPJRDsrGADK4NPMkFwmnwOr/f8dzBZvYdPjpLp/P2+JB5EkMSP7l2Rjo4RX5HwlgrNnGHrD/L72h8pVrswdsCsCzk69Dx2XZc+3KNQ2T9hHqY3P78hoIEPKqR+VbRZJTVq4eHeHW29WGNOtUddCtYoQgBWOBsCdtmLYOS+aomOLJQwpYW5u45+XmD5fFW2HZ0BuVCr6RdJp1/IQJrWyRDZjcE4x4+OMuBqWOP+bt8bk3k0ZxlaYxlg8cXNEmnjSP0otMHqCR+U+AbnXiq7c99a6R6QGbM+MnlPIZUbO1i+YB7ySD3Gn8E3yF8KWuBjDhIuGlbXgpcjXpqNzZV6Vkwo6IkBpSGNPHeTyESyQjiJpCcyq5D4fwS4DyqJXZF7mE8Ag3+lnAkqlGgbKXShcBnGQbJWNrgE9ywHFhfXX1KIpT42bUkDtiGa04fZbnwRhMw8rGl2Pw2PyLLntW4ahqnGq08fUu53m7xewZNRd1SgPHnEOakMUXBQh8i8AvOI7AU5FJXFfXV0R8vxSqpAWSf9nImLM4D1qkSXQbk8ddnOovIqgqwHnq4UntFm3c/Y/EP4DKX9VdHfL9aT5K4nXGDVaBb34wgJlS0jt4Gz1WxJw+lIYHZ1jn8+6Oj9Vq8hJbSGwS0rmBjSdWdEpugqwvTFfKxo6SIDbJPu0g49E5pP2IqyjdWlFHk30zLYh1ELnqQu+Zp/QGUB2PmqaPsa30M4IDwEz5pq2NqtrZptYvOXQFRNV2RdXgcgWMrMzVhMh9cJr6gOVQd1mSWt6Y8HpydxgQfLN9HNW5EcQ3O7jRi2xx1V2wbJO8sIQjqgLkPIGrs0SyFSEvoeSQGAtTOt+zJpUGlA+DdZzgrBvMEzgOpRuxqQwqtBd3Z9EXuQk2xVhwo2GDxJCAcM1c/kbjrqM7Ea6Z6gZJYxJ7u2aGwnSW7OvGdOjsOl+Vch72MS7dNU0I4D8ozwfBFqsGRTmmxR3Bzx1lBdTf4Y5nnhEJNdjPXtWJDyLgcS3WzvDXmDlVaUnBOIaHCMqB5HNJCDIGNX4sTFhjFZEjtkwFhjMbhKxxb9K6H2qnrS2TSXEExWXw/sAAZ4tFAd1bPpdqyJOzDn+0W4zpDDrHIgfYH8F+AvV3zfhOm/2PrxVCYs7nZcF7NxDNKaB7a4n3QSvHoHTRbkb3KG2ftwpg95y8Bl7sb+BXUcALsV4F+b1Qv4RKMFLvlX/+0nSoYCLiHNe0PEj25mHCyQ9RxE3CIkNXvGAQ5Yd0xwr2XVp4nl+6KVCvCYtNXYD7s6eICnX3bbMNXfOuJ+gE58hmtNqeOCKzvsa3ZdINZ0pWvVZpU+vnOTJVRyOKSHF/xH2lPLpfGgt/dz+NGqHV3mJ1JX4FbFcPvYhAhWwkzRWEUfXbMDzhVdN9UbfMCV97lh9RHXMvUCL3wdXYZEiHb9O+kWuaN/VpF2n2s/KCK+jEHKU7LcxWwJVlH2JoN85QUfIsxQNSIRP0QJmgzS1pHWGxO6eLq5Ll9N1R+OnwF0pRHLgSuaTSoX3uHtMmGFXoH9X8k1cNHOTHoBbaElD19lDM2rQlAAOrQ0i4P2cuEJbhmKzz4LbWFV0X5f17MADdg9HG7FRBLXxuzEbkmzt5zgf/bQi7TF0nWBOQPQRixw8IS5NDUIbCl+UbqiNH0Us+JVpDSVZ4QMrkIqAojWizVGlYgp8J8OoZMj7Iwbg2fgNi9pehNJE+AghxhDOUEr5pHI79dt8iGO+Bvd85XWr75ZwXJa4A2oHd5vOP3xoPwGvJ8ieOHCbmHoxTx2T9t2PqHtRrvymVfbP7TsaZLtIQT0DF2v3lVvnfAWU8h9A6wzjBxLK2KQ/eqYkiSGg5e6pYmnXWgV31VAtq9hLKKq7hgMjdf0XDQqH7H9c/xkQAN5/IftJBAExAvZWhfcRAP/RKfr+ZzH5cXQBKfBldBuHKe8X6kEcsR5ASPH5CPs8GqOJVzYNyhbkFhVfZtzBWK+v3R3Du6f4rT9o35BC7vwvVvNwhZh4u2m71T7e2ltSfjpOyxp3WS+TtfUqGC8OA/OXxLAW0TXVwq+xVXf7jQjmV5A/kTkBFYhFPJKYHXLqzFZwMVe/SUIqWEhyeNYTmuaSKVZjwMY2wHoEeAKN6hHuLUcGaeRSiP8/RSr2f7Ud9RlQBb1y9ayfS5f88oXJZrUH3jVl5cWMt6eOWPe4M7B7Zqwf4+c0tWYsMGxvZv6N0U/EUZfWT/WVu6XAG8zmDLrKppOrFySaZcm0D12MpRCumGGb9YIx3+hBIWTM0h8HHu7vl1UvK1uCSeYIWHk8jItDD+VJLVEgUlqI0SPiTb76dffHWClJuPeZ/B/wBb8BVcIqFdQ6u+fVtcBwNy3f5xgEDPxUXQWWPubkNZVPDUhhsrAJSsdMuJTcL5LA/ngvzvPYnjqnXoibFp2/7dLMd00WGUVrcFZbIKxk2OU8rfxpC+Etn5jFnDEIAJPnMUBzO7C1tB4VL+KsaUu68hn6Ppw3xGwNs2Bt0LIH/VY+6wtJNwRX3Z+O9Wmrvfb1xgSEq4dE7uQsfPcaH2bR+/vE8pDSt2rGU2JHhRShu6lNRmdxPOk9XmGDjx6Ko8VcvV5ZZ4IQHz+J+vk2OWa2yY5Bm6cIQyb+H9yM8Efm9Efu9sE32NB9OQMRe7Nxj3vHsFSRaDbMAf5COZQpl9+hwbMRpSDUWaVejl6PSs1MPNFbwCLAfPKEnjJ/q8yvA7FB/xzjT5uDmR6tDU5NMSgETi8/lexPs7zwFW+1b2aqHm0Y7VPGsxeGOsRqfE0g+Oyrm5aI1dOvbS0U5AGNlo7jndlvbY+ALSb2JbQdBtKtRDeUX8f9T2W4yXv1ZEUmvo4K5SOgDuVu2Y+TcmxY3tJN8O0iphpMhMnG6xHW5MjJCoyGqVxDWEntoMBHfNB1cuCL5m4g5B6Z4OBRV97vk5HYbEu0aDR/GR6aefPeYN4ynkuRAyRPLKBV+I8Yj9q5aYwY2rFijbJiAxxLVDXEw5qtjnZupYgOj3CP79Z7ygk0Fch5OZoe9+9bK7bHUOjNtbfHRobYaUfPJ6nLoacF7SxFMAJhS0WbbgpcAGvmUcNj5uYdVvr6nymRjaVn1g3j1m7tlzSbvGxU3gcF7IaW+m9Rau52SxrEjnFLdD3f6s+gg6vHMNfWiwxdmqM5EDhoG8BE5XxMqzQwZWnn6Ew4GGyg6OZDeyzPIyCDyX0+GDgqVNUb+hNzFcyNaj0Jhd+4ZrqhQ3iQxeY51SFoVUKC8HQo5vGIF8AUWPcv8oMyPgrL0nqYdMP/8x+N4yhddK5S4dkNqwwqfe6WlCQR43vw/xhOaKpkTylD7FbfQdRkT5PsxOWBAPcZd3JpYNWoXxpmEwvhEQUNcG8fY+LtNyP7mbh8o0G0QlQ1svnkQBH3RIffhB60GZOvyoqeaz59VtDBtDJ4ZutaSzsBGUDJchbN4Q8FtZpqMFjMljRYdqXJmjfb1kRHIX6+YDlhh/CCrfbfmjOEB8pbB8o7xVR+8nWeNL5kTSjaZ94V+MOYUJtrRgRndn6j1n9FpnFzWbsO+T5sUdN7RVzQ5nyZkoDgcG9Cr2c+SNZddjRvSFGhoWhlfr5mcdEz7NAMRUzIr/1cOfffjkzsFyd5YWs6cPFPp1B/o2Am1OP0NY+qmkmkJPsbQZe7SQ7l3SyeYoRqmLhHGCtzXx4rJqJs0nxTQ4SE+ebO9D9Y8xUZAPhBeT0Fud0RdRXcI+AIH3JG5ddOYQeIywXe6CiVgbZkk+8bjZsmVyZStxxTRmq9fnJf3mYDWvnTJhZbdF7IowwbkLR2Xc/Sn2NU4jYV/3GQpk/lEtdDZi27fM3lXBvflHfZ1XyO4W1jVvG+YeSIntO6b1oP26bqh5/Ja4fzAWU0IaBMSSTl3rMPn/7kSGrt/3jTGUMhUvPTK91VEsEJhf34OHcDHtDVwxb+/KwiTPapd3JX+HXG+NVq2FJYHW+mZ9dpR4AvaZ7gQm6hx3SOddq1W6Scc9ekLfPuxyjvGwxgNpGOWAyBF6nEnSDhzAlGvMubwzBsRG+HZTpvSmNU4Q28ar6H1Ptk4cHlHyU4egT4BkFxrXJNGWCN8VB/uI9BfMbjhGkyR3YR2Ai2Fm7k4Bfdbdqc6qg3DvlqKyHDUGWamMHpFNIUuD6OHoLL4uBgoUXjjy9J0OqDe/H/1KeDUnKxJbn9EG8nyO/U9UaJM8bH0fAXX8AGQ+WN0hF9dGMNlNGRg2HRgEDnTv/0AQ2XiOLedL3zs2QlMiGS3B8ddBE6kcs2uYmTUWI9CmLmJB7TgHMB1JWsbG+p9P7j6bLjlYRdK8mqI1nYHWbdvaRr4Kt3vhc1WZ7ZaJgs4uBRJKoFJs5kNUFwbcDSSFx2oL8iBPkR9kp6nqZIuWrNnsSApi4pQ4+Ak56cG0+kzGkOCc1EUO0eT3k6Jre7Mu1e9+ktXEN4CntYE8QiwMoGnHuhaHV+O8Wt5V2hsUfUsSo0f3qpkwKL2h3o0b7xwGGUdPhgEUIgxcXPxIBo/nrmNjybSbcgcnUAeR1xEZe8hxAUI/AcuIqBBH46YZ3hbHzBL4j8pQ3lK0YOjb4MHwTEVZ5ofUHwdmo6mtmEfmb3r39BtoI34QXaGUGR0BgWrrQxhHgZxEh8WoVOFNbdP28Qv4uv2OSnkWXLP4C7bcmE06dKS3uWpAv9tH6jrmXLW6P0gSMzuCLZbwApeq0GGPMSTBXfMDzj9Z+N7pdbA36OBLIk+4DqdBTrU8zDUzZLk2SxDpP5Ui2J9P7K1LBCMXeNlydrEsUyCr1YdGvMd44A7Qe0+bZVjY8NDtMuRnMRGhMZQwD3vKmRBKL+HYITJI0oNu7jgVZ0jcdIfkNN9ILGGy4VvmXh4WhkcqCrBPjCqGDCj+MOyPfvQ+MvWF1IOBjC+LBkASohVixV3OIjOJnCcAbu1jaq8zGCCLn1XvI8NnfK9pDbpD8MGYIQoZB9xrjLNYuVbiVUu6+tqNGQTTU5UwbtgYAXdTEmb02N4Fbn2lXAXUmqJN3WZVbgMi1w79pTe3evYZWj0J8fGjhyjkwoMxSDmaPcp2/cNbY+n5xVVjZ5RQk+UjLGvxyYLZVzWBwnfjeYYwrcrQnqYMcCq0J07Fr+n1SnykJNrXtMQx12sMLuIzhPyzEjHHH0Kgq+lXw5QJTnmFBJA2+q92Cg2OqkK9Cif1iEr3CUXcHHrJc0XLQFvlsi6ON2wwX+TKbn6ROJbAKYzvSyFqOJRVEZsbgHmEYoVy/tIoNJSX9E/h2r9Z4ALOxO3oKrALmzw+FFeu7Fgd6QOklsGdm6PFKPnXrTMG1n1uW3e8I4a6f0IcBHGjt0txhHHwe2IqpoWBRLVkD5vzmOhWNcT3YV4qfv7BtqK3jQBlJhyXhnEEoQ+wCUDUDBlEBFF+ivF3VtEqYsFg1zPw6g0S3g9tVINC9ezrXvwIAeWhXFaT4s+uyK5qNQdkOBYfI8WoC76aa/xvyJ+AGuR8qs4ySE1PqaB9LXI3kOIQ2oqk9ue8yW1kbEwwP3j/uIGqHe3xwh6GYdMVd5G9Kac3OmhEqEiVT6yHhBbD0SKvUbrgKnoeQ8zq7H1NEE9Htr5e3zLDzUlEJ7LyRKvjfzJSKeGKvsG8vWDRqyYStE/UnB+3hy3FZCQwRAthKuUJgoIaD82Qrmxaj9mVbD7N6GJ7bHPPJl3Y3BPHkGZT7rA7DxOhsewDxvdv3dGjo8xvd6JUfJ6Sx9eZgsX7GAVPZ1Qo2rD0ePh1uRaJwHJWBjo+w9vxSFT6g3BRqyS3ZIU0KC+X8BpMq0YkOFtgNvUSSBCiUWkSXsNZkJ8A1pv09Jr/v7MBBg2Vl6YVB4LAcLc0yLuUGKaX0DtVqg2nqBpxgGo7nmj+9fLSTmOo4QT4rnyrq6SGqVgnm5a6wWAzVsvDc2+W1TNo2CdQzg8xn/5yp7NPhiMOJ4K70A2U2MYU9Qd3b0TrXI1AyOtEzuBr0b3iUGZWYHtv4D0AjXl4JwRHLkcW1gS3IqNJnDwuyAfvZN4pN6U61S7TDOiVnIVqahjXzvCW5gjuNNYT17xaaFnChNPhyqtkcFxL0zFsMqjcz2IoJfuS7hhYmdZgoVTCCS8zAZYcUSHUzBdIwHl6RTXxZXEyLyJ8gRGhiE9u4DU2EI7S0SHnkUgCTtqPgXovkBTDQk6XTTd3CtihhdEZdR5bmW2S4k69VtCFYX9dI8FWUtt6zjo4LcsiMHpUOQ0wVnElyEW2WTvcYSMpfeQe7gQ+kUaGZOuv1IwCKSlZKxf0WvZb8G66z7i0TnsPKwLehZZ9MJv3MSgr16T+38kczK+6Z6P2k2MSvtgJIihp2rRbS3XtN4O6OsAP9pbIgmuVQRVdkw1Msp4r/J5hCc7lL4vVczQCEDZFHjeOVlnVDzhnT7iuoNOoh9+5vNNpjH9kh3sgs2oivCw+bbcvjXYBJj7RZmVSfWvExyeYa/3aqAPsqQ2zkVXDHdhDL4l1JkeNq5NuuULe/5P+4w1n18LiFr5PRQM+/8fFqXiHh9WFROG4XyxPhwuuCEB4pqaxbytbwrKpMYXn8SNJJSiyv5aQcQaI1yGSZiwj0N9n7vrFZUKPFxYjjsMM5d1HtiSmvNh9vEYtrdJ9QLbEJchImj4WFvCfQtSPlx0ZmrC75i5v6l60IdV5Svs2J2gOrRU5X3DWcVTnpRKCkYVOzPZP+d2W0P9ZgEq5b2TDJmFPeN5uWi3K2llhbZKC1WCdSROlXC40snp5nzwUn0j8bYPzRN9hBKMPbjF4gymSjXqsmRnbxdyiSWNjnHNbXyAq6kivnRwMr9N5NA7oqGNnzXZ24f3YtmAcYn1n3bQftwlYx5Mys3gnQMqoJjbngrV0ML1RJKJ4/iDGMMwQR6s2eKMNgrTb9V4W/w5VoL6f2KRSN9DDuuG9IP6Lnl/QNOSlh7KFPGCZIGQKv/Gf6HdNyZ/FNH08a9Kb7khyKyL4CVoMKjW3x0lwpOw4IxqyswvHbbfUvkyYO8I4hIufFtKZ3EJB6oZ1iG8nwvuy1phwojbi2T8+0pubYhXpSDgGVNDpaFZkkWgt7O5AbXrzv+ELiHO34dRu/cqZYPOqhoflRnhV7iD1vhj6bN3TgVjCkLaPwvS1SLKm+xl4J9jzotr5EUmZ9tQL8Ct79PfcFWYhVQvO7VrOFQVY3Vd/vAhdp1C0rdm5z3i9rMp/mQqHp2UTT5RO2YDFiACPAGNHEazkwjMFuVGoQOe60cDKDRl+jf8Nxcy4UfmxP9dEKEuwWmBiKtV5KiFzxSF4xWlK6FtYbsRg7C+Nk64sFU5yFaTObL0pDytiVwEiSPpT7AQ53lmMuISVDnrGcBqVfxkjXAlmd0iIBS0oWeTt7ixXLMC/VP1g/T5B9JB8TdgYAr2rklu7dpH5g92xcPNP/HfsjJT+Mhb+Qz2n/fA8dOd7/DhF0V7Y3xd8ry+39DZmAMsLRxlPCOFQqVDSzjFhWlVgmSsFgWwlFAMA7q7rcazHfRu9zPGSkYBwHTLYnJ1Oo7UoJJO6hmiRyCWgqH6M6gkFjHvybTtZAaH+B0PIrf/LwQMHJYCTZSnpCbrMSHgiwYBZbq4bzaMOMjVx9/EXF3Ru5Ur/ZH5K0ARqjowAsrTytlRAFVnlhXHmjU+NqrGw3PTfXQ0SkR8Lr0DHlnk8gnWl3L2Q9mbxlmxFWIqSp6izR6nmaI6f8eC6PtNtjpK/EipW1tDvXupvTPUyXDZPVxbyCGr2Ygm/Hkm/EcGDjZVWvkPpYKSEhj2xJbgl0H+hFvYGyEHwWBcPXSwxPXuyItA23jMp1WY7/6IyGIbrHW4XUxaKgeDmKMqyWhqOuszFoXx+xbAIRaXakTQh3bCfxCI0zdtu9cY7ZfsiUyDKCxP/HeUQa+zhVdDvohUC7oJHCOZdJYvlDYmT5UrvzMAJCEBxxTF7k3DPQV8OoOHf2BZS2QrpAmq5XvzLJ1gPFlxnX59FGpxYLnyhK5pYa4MkOmPUMvyW7rC0Bmo4kwgy2/y73WEoULODLd7wSMXDUf0ieV9Ic+gu6BpbBFXEokuBhBOwOCe2lGbU60Mc/Z6fpyvT4T2EBDM7cPwkRc1Mhxqix/1A8SnFhgB8LOzSHt3Nuab4VJAxWnjVBLV5NCX3chAw+XukXXLzkU7oLldmtC+9jSJHZtGTVnYTuTzS8ycJ/Fw1NBr4h4b1XX9dXUBbOxRpkWjWafugs9VRQxYnfQkyyWHyHt1ROkTnGumLpKzmNppG5dS6G9zO7SXO/tUmgbXxTEf5aj4Ksr4ApQQT3cwVtZEzhkvGhVrmVKDn7prrgPUsFEwqPZPdc79KBCYlOIXMZomnSk4Jt6uR0xrGHsKAYe7pxNe3jBHtKOZ0Mdd0rT8FXygzJt+8qDwYQL5W6Qyb12FNfUNTKNKw/ZmIRQCgSPHKAhOlkWWxGn3XT5wltbIUjTCHxj/6xAXiyIvXFV6xRSjXrtnGcVZN0TuUdSgkWlMY0SPIYCLTvRgBz+UJgBA2nw0rGKeTwTOzqzBkbt6NyQM5PPEaIAq4AZ1Ckb36RTH0UCIJRFs8h3UWuRaXdNz8SrMqQ8QT3O2ZXDsnqB7lSNsbhpwL0NhVVjAXqiC+A8yMr4hzENfnGK0l1UvxjAk7s9EyvcxUYpH/G7lAtTajWL4hzt8XyQXFYTZYzUkbvuz2pHCPfNEBtEcqPwZBZdsR0Hpz7HzTW0pNCR4sdCO1gW4e2VzkGHStDFbTTzlZgAK9STbzzyKT4gfbClrzfGt8EphUi3I0V51TXvvBjyZ6qkl7jb1nN9uuqLR5l08OMVWfF9N9ZuUHYDX3hGvapdUUXx7EaiEhyBbJGQFc1ZHM4BbHRKbZsUc3vnUJhuzHRYkyAWZh1sOqnJHd5tWgf/GiQhMXdKOLQNyRCzhcD1VhT0dWKXZ2Q3g5QdTPHMm/bjnBgq+sf6wjLiRTm8B5qi6HHt/0x5mnXFfdROnW8wuNGCkd3fnN9c6lM9Hf1CQIL/GN1wJV3+lb2ffubYaHrFQgMVyHGeD7T5sLDnlfR4Pn+A8eJmxMzhdyFfmVVCTsUwmeVzd9k7HQBiCJmlMDn1nNo2q+5fFmPb8wAExwuAutvdKH/L6NK6bFxKRFMDXmZj9Cxqnx3W+XSGdvcpO9ITJeZ7FQXEgmEJ6bRG6hTZDyDVdqlXdYjG9vOy8f8+yRd1SpuSG8I5u6B6ZUr+2GvJFuumkIbGvAYFwbiRCQfwbCnpy8Me3u3z3J1E6rUQFu3AHMhYQu8AlnyNY4FNoPKKbYn1z8tjZIGqnjNlb6xbqyDROT9KmSLNBX4Km5FEkbPy1R8s6cRy0lhqVvtFz5t7J9nvkxwWpKSePGKPYEtKvXoyjJvrHU8e4MKg/i8MZNb5jhK5PtXIDsIMlzS1pqBX2SiqhYpLjvZtVj25y+iPqCE9wiVvKQ7bFGAu0uBv8b6YLVQ/NsVo9sRX7JfFsGJ1ICvftBgH7OTsdBIEvBlyZdnj7ri7a3U9WyEi3Cwk+yq1t9SQIhuPFOa4kVrMrjsK0EYlRsTtxfVCe5RvAAVkUh3xg0S9R3NBakN4KrUZBWHuzWwdvFBcaO9yVx9O56TCTaS9ayErYgr4rb2OTnErR/yJ3dtVLhbHSpYGlKq+7YK3uIDN4HBqeDLYWr7M7R3vsJYwzoUUu9W6DDiFwOEjBMG3H1MgGaFNh2eXdLHa4Wq9htY1D3HjTZNZ9vOHeAgXB5mgAKe5B7SUX4tzEw38PFdOIhiZTHrEtUbOtPj7OxKD4N7IP0iOyTNsu8FLNxDf+wbzVRxGLiNMXQk0bCxyrGn5TePBR5rBnxvpdYdlaOcpcyRwAl6Y2C0HpGM8VKLnW6Rp2MeorEZb+A1MRNRLzUoZbjUqCOmtYAWBc04TRLOG6f2ZVqEi1FTpjHPUnTl0vHgHpHLk49kGtwCRSFQPrDbTJUD9uOPWwbGrqFBY8oI2JxPvhyB6EGv8z/XnICz3NJQnJHBCJpRCrdG7cVILkcjmRLuihH0iErHuc9o3CyV95KVYKHCBvHFmEZjP5VHG3AdLPp5T3JpU/gWjjx0H8l5SRJj8K7DDzq5JXYgBnW1Ha5UxGhZRx7gS8VqI2bTovY4+q8LcOJYbWbUr4Jvq7vAVLwGd+2jsFRGLDDvUEuvIpYrx9BX46mqE+kfJXQXJg3B/LDsLAWXsYk3+dAlvEQowbOEU2bFyQyyk/yU4cOAjdMb+DNNr0DP0MCJNWzrg62rNQqK5jQvTCzDhGyQpgYSkRBIdYmXwXlL8Sgl0WBVIKkq+SBtOFj4GUAEcwlV0OwyTkVjrQGmyhM+IM3rbj0Onu6LX6LODPMJiRkp4JPGOYfu0qEtuG7KdLJlYy7WgLnTEdcd8X6/uPkMVZ+FlVGmQbeLZW5NPVTKwDPkP7jhNORMXZxOZK7aJ95Uozpb4sMe0lHkF4qdyDvQ5aVjEg6gEPPLdm3YKBmGPj/B02zWrYcrM1WNQfJ/pQh3QZb+4gI0Jn3mGAUOHeE4R9aVDf3uYHy7GGpGsEnbDkY9mBNUjm5zd9oM8WPsaKYaZzA7E+pfoh2dG52DPfRbdbbEP/+3s1awNX9sm5WVnhXU56Uqb0vrQTMJF+XbZ0W3cq+Ma1oxnaGtaDqHOZuVGZdx/yIfIXOTcgoaW4YRxgqbmWmPx4ffFuB6YFY3NPBrvLuwm8Y+ecYQxUhKsBOJTLwHOBPfoHJ8TxSLXcBqIw6Bu6D2brNDIDrz5BSmHChT3X4jZo2b4B6QFHMOMQ60TjKJlJs2O1YEyZQmw3zzBYw6wLUMXGuOpkcPVvVSrrD6v0BdE8jggKx6R7OytQkI9LKjxrOD8Ty6KuCS9zCTMIxbRLEtDUwGcjVzj7p/XYnUI8+f8rU0h1ZvQ3dVdjESM4T1rBgsYXnByfR0d+MFLvEWnkHxiPfjuKVyiRt0hGd/bDiqbnhBPIFE1TXMz8knGeHjmKsBO6b54J6wg3vhiSbWRPRPjjudQ7w0WYiLCx8FvCtV7fUoZO2Fz7+Nl00JdLwmeGPvC7Ux5KLQKk/9OXZpIH6MEYZ7CIzrRiGIxY2ZaqzMhGPLWrmYaXYbxHl3RMRBHrS2qGCUGB2ATMbFuTpvYoGE5Qb8xkGTaGZyQfwEh69ttnqgwAPcZjPLXUhbzuVTPsmw2AZewjFQYCTSDK/bT+ZWq4xSe38gCXaxtqXI9Q/pGmLadQ7lchbDhEwG0ilyfA63RbWzpoEdbYNFfSGlRlUzxO/7NzuxjnEhCSDkM5VNrrBMlrqsIgK2jDxD2k+sAxCyRd+Mr3mUR1Hj8KyNC2EFrFKdZvD0ZJvOWyxrLMbpLSy/p2ANUsjXJkLaPh9yB58pUwFnHuj27QiUj+lvKvRkldrlbCgOKKT+IDW+UrgjCIWvDXOPX3HJaakRgm6zNDhBbMLrBZjLXq0ITIJrtjRqFuB1TwxvMbiUD2YjGrXV2EZamtiJNvzIbf6EMEp4Xm+qxvcjwkc/CFZqvtoA/hYbH0Ppz2x9U9e5BkqvWKcBJJhMyZl3T1Q/NO8MJQmBN4VfWHTqZAHs6VjahSPYWMqSXgr3/S2j/+j8eNaxcYfMejtU6sHohPnH8jCgk6iwN0jmvpWWi85/AWqbX8YukUkKVkdnLn20HmGDDVBkxh2Alp6Vqv6CQGb34JSTmyvYoBPZQRXk0EJeXtNGo1OJKEiTo6tUx1EpymnxRpfLbrQk69Kn4h3OqZHZrHDSTWYsqtFhp6AwNw/0GOQobc/J2kYuzPJBhE1TqTKaX5+jRZqqKMhB7jB1zMaMeaQvrnfv9JiKCuD3hiTYMsvDsN9HBMhXQ1umtcUmNTq9HQhvrnLvRRDwIEeOlnC1uFYjmAxIu9kVb3UlsNYjT7uS7b4aTdn3AYHMqsrSrpTkJeL04uyQNWWBdTmVoHLxwpShzR07pPv1JmlXbRUxdDdGZjHCR/MgrvlF1CFMS7qJAEAb8oB+UvGK6XxsqU9f7fOdS5G1/Kt9PQewkSGrYMXzqSjgCCdb6ZP4yQyGtBrxdxC3kqUTlYZl57RRBAhCEl/xx0YYPKzX5XHXSdTd3SINjd5G0kjvH3B1AyCDEAkJbgH1wgo6x3YwEvXtm37BU7DzPtnL2g6GuKwPijmBYKyGGNYCrmN9hvWAClyFzFtXQFF5kpKnaRUVGRnsQyG6WDRoMVjrABRRajBK3Yy25YhIRJwJsqsnLb5dx18nmPVNQ9Pd2nqVtAE25jMOMZnoIvP8FSTLlhwEsDf3eU3SMvsUjDVEs8mCApyxv/wpbR1XvAZ/kmVqXUtShyqcaDr0rFAjwDNRyUdUfswJVmMz2mHqIgYZzaJbk+0szvMSS8wg7LXcWsDVBC0ZBNfumjHjkPRTBCzBKA2ovAKCvVh1/0aqgUTqCtD8G1/mB2c1ZU20o9UmfC85VaH8ZzcYR5sRL3LhcAr4FwRTpbeYL2HjH+5vgLT9NbcfwXxUbEiBR5wfRTIhq2kxuk70hKOTGdOvO3OoU4EaXLlXN3cFvx7mrLt59m99sTCkCCl/dcpGqmh7AkEgnGXTp5eY5B5e8ggZyRhjgVROGlCDtCz8nqD+Ud3VbFZoklZR5AU/sZmhfjPUWgUkRdSmh18YEjCXaXB0r59+4gSI7NOQULOQzFr46skyzQL6IffmlUYgpmYSrNnrOQwM2mmX/J1u0R+xDZcZ1ULH6ji+wpz7Ug9XGi6o7M/e10ciUm5YE0CUGLfXo3T7XBaYGRmDbjvHMdeEH68M5SzeUxoxcohZ6Tlmq9m8Qy+/IF4HNL3SqE7BVoSAj/MiXFBhKTtjhU4xqXrLVEJzZ5MG8afbl0sNXVR62rtb3x9nKqcAc+AH5P3QWy4yVVsyeVG3SmN9iWEbqg61fb3/X5vuv5a5hk6o9zQfUfQR8E54ChV2ykF87VRa84vixMZO5bDQeT7ILDIg3U3E6X+7UkIuoZYb4bLsF/KqQXpJHK+Z6Fz25jP58gqSjcX6H8xVsU6QF4/8+CQuAyKP1LHvV6N5z0qXfMKhgpDcEkNc0/SPAp+tkfwyYlFxGhmVZhdNTQAgJYKLl4UoWI0dbnmwu5Ypvhhg7Db8KXmwylGaCGkzP/ZnuXmfW8xWYTDBW34V3mMxjqYQSoZqamf0qtjRaNdqjPpAK6WM6DDZj2gSw5g0H80pqvRWv0Ffa4qyCpGxkHdwhgqpfIOZd55V3WIuHJh5FKjFYdqvFDNBlXO/ODwKBtJpKWYRvnQNMcX7UNF2k8tIu8nl4M4z8xBRAGk4jCPbJBG8udt5BwiphW1aRUSe+KsWvfWXvqQ2dtxK2z/GSMxIMKEcrkYaIj9Nxu0QO3pbwuDX0zEnhZ4sLTQfzXLCYHCqpQY6I2e3XOCAfkDcYsVPpJY9SUGmbj7Jm+SR3X2hT9Ii5Otd4zZhRlnIWDO8uNcxuD+3CVvVg8R3UYfxdb7uNJzcCRCGa2D37bq6SHkmG4bMF0zkEwCY8M33lAoQQMRAQcp/1D5ZDDEYOq9V7yA8mfn25Kt/g2BmWyv43XC7Xw34Ih/AzTDuZz4tgP+bFz1ci+xYZJmJiKeHIu8B8V9j7jzSS1mTBoP6i9oPX6i3dvF9BrkxfvgRZCUfx7Y26RiCHDv9zbHq/CbgyIKmzKHivIcfSzTN+5Po9pF2CPRD5F26YUiEK76TWL7iRC7gnxJ4b9uY3lHZ56PBdPKrQpDh3L92oZxjvqpsXk7OSDFvXR6S1nptsiOpu4z3cXD9n6OxFhTaETyqz30b5VRWFS2/ffvM8b+5ydVBGLXXkjFyGK4H/vABoyiVdyAtHN85Z/VwI8HWcTw7KLsDkx66NGOFsnpSmcCMoCj73g37H/81cMeXdeEC8mSaNb0tnxuY+o9k47+ktEZ31Tb3yCz8QP4g+aN8Af0CmzpWQsGe/C6sHxcvvlWk9B9L9aSbOA0IFa9dcCqq/Ec4yi9uyDObMr1ydNQ0eIhCEzMKDrKmZ2MN3nxv6j+sDAzPjzfMQKp8oOHuVyNIhpqjcUAP4BVLK6hLS5BJ3h97tCYXuxRYqmP4sj6cm5A3jN47XFaKx/K0uhzDu63pl0nTsx/zoLH5Vc9aqvq43HlnZLJgBFvlZ1H7KoJ5T7N4l4fnb5GL95tS5zxrdUn/ZI1yNvRu4P5oUmLoiX7W0/6UUn6mubJjr/o0B6lM6lq0N+l9+RLl9l1btb9PfHOt1VyYrX/P/A4kWz0LY/SB2XOVsPTWgdOh+nOZgjuwaQGnW7Dbzbz4bk6x7Fbh1D3nfK2ObVJ7Kc/IPaW3T4h+FLW0gnJ0lMH04MsHPV1MBxcmBp3tHTMtz1ixr8njHX0cS0wYzCBDtA3XG4LClY/mR5nzn75Yu/GU2o5HhNVP/fqYrNfNiJzxGL8ew5nKgcGEVhMUR5BDR8uAuaZp4C68qdxAshEppLfwjMuDqnGmmgoOUbtsg2YEpxZqJkFEfMGh0jUcpdMhNBpxG58+ZpDPNe6n72bWC2nywrtTALp/ky/f4Q0ChpLjegPphE7kSQ7aVOmF9OiV+NDXJbtNk7sxR9lDEoRFwOH7JiHBE4ZGLvO9shqjmXZmVEWW5umLsBXpBYt3Q+K2C26Z5hbDv6yM0lvMlGgBIP9ep5lLzmIXAsCDuw1C03A2XQ2EzGyHC7s1pbOd4uOdK/hWORwEXqf+nEcrQ5FYQ4IVdXqdBBL8Hk48HvNaZMuG7JDAV9o7nAzvYbRGYlAKaUqZYpZ0AjqjnJ6Ub1dYOcbwEeD0pQpnhK5/3CQmu6UqrXPaUQcLgmmTGh0oDF8YnDMUzFJ8gR7B9nSs4tO+8KJjADnxuEI0LcjE3FFN43uKaeGiwf+o0s2FhgNfqL9qzQbFgFFUodjA6EnTrBDQBN+PdWqcLGhZXDjOnyg5W0E6Qj/UD3G9NOc0bZNijPu48HNxvOttSx5bQVUXmefp6ZpuGW9DvJOBgPVL1ICDf8gOFuoO3QS+m2QMOVO4OXNzWnqHnSuAVcPdpcU7a8Tm+EJRtsXfnwAy+vo3EP82gZcxdI/vzEf+u3ftDwxeuWF5okMU9mNYx5o44gH0ORXI8H+pEQwmUoA7ZmZ/J9Y6iE6r5bEVYb8gG4t9vhbfE+/8l+S3lCnLQGRtdoFZSu5nW6sQYlYqtdgzuPN3TvE4qM/hIVFtSUaEf+aStzKPR8nxYdbbOz5BR8UJJmdVaj69mhHCZRaqE+hPiXYWRYZQJHF97U4T8GxC0D5k5P0+fTn6hwUzMDM/ff2XOGdmiwgOsVCUDl3jzhiXGU98+mD2zHOCcmOHn3r7sKcWKnQhJYCyHmJGw3NnthU2AR7Yp7ph7U6BgLnr8HFZzkDP3E8yE4ggP6Fp9LCwtzdTWv/LMMdNsh1RNddU2rOWE+U5aSiqqTXqBvCnVcQt1e+JvA3Vrm3wKU3Afo6TKFZCyeYm0GAMYf7eYkcGHPmbCVzhHaZyv7mRuSyALNCY0j8RNB1d6GpC7o3H+OVjchLEXB5lEqMO7V61ecNgfkL9bHHWexgtsNRwpT0NvZIyggUxqt54aBTtDlrb9w6ZfC9gLZLv26MsLlkxgPx5QcNohWCapvOZgMfRDGdBm3lO1eIdHPJ5cG5O4sm0zBsJcRzUcqKP3fMSV6mb5snDC4j+4eSPMBk5FTgTU1Iz8Qpk+tZMcaZMbxATHdDF5q9c5dxNy7ZoW7qf1CQiHJ3q8YhDyz7YdXoyukwWHvqakx3BzrLu2jENGahPWgA2hI5aUnGsqoN8ZyNJp7djL/wyVedg8d5ls4Hsz2oQMBhjeO14Edw4fj4bdLkIvPJ27P6JqsOgtbqWy8YXpVKhBPrjeQOFi2S5e/sYBDTH7M5s4eGdMeKOzdBiZrDDQtwUTbhQTp7e2szrlQaIN52dEsldBNoXG+9bDhrjjGiymTHtqIjrW4yaYkLKhZ1QxL5IN4w2N1GFjpkH5ZcMEGjnx3qGBNchzZ8OCTgEqQf7NRS16T3GaLmK7bCUtjd0mzdhbb9aozAzCcq+gMOnr4IyVtCgvBHYEwbRNPw1Omh9OP96fT7s2pRMcrmKbgZ7DDm9QWX5LgtpJLcIpEyHdeV2MvvInm+l2f9s9x1l43BFQaLUKETzWKAz1gMp3t8Dd+IxUCRcJUTGr7rrczSnvh7wG+UXp9UqnDrxhAZJVsRbgmOCk2SNwZ6OegY5JImN92FFQhR3HjVWvdnOPUvYxN58XqwrP9uhnXXqrs+HEFxWpPIHdbOPWYxRUAHVXtSnUTAEqmuI+WH0ZAr9gDyLXv7ntF7ht0+5ldsQDVkF1GkQ0KpkvZHwIRo2ktN1fh87xwNO5exmtWzqhDB4rQmUQ6LJxzRV9bKLTUXOe4GbgkbJtrPW8d7LhaAHfrvwwmKlyf00ESbpRnpU+bR+pdBLa7eUzntwdR+RmFnuaSHqHw9ltozXGU8wqBFg+ZzxXWAfRGz4Ef5x9iJaqVnnqxZ9A+dpmyh4h0cGtZc9cGA5WDyLt9pQgBG7uSYUFVGZB+Iq3U5gLd4X3I6aaI5tmWnVZ2rMGy4pnQwnLhtXsSfABHVcFS9o9AdZBGTiQFtKvWyg9b29ITDszlrdDaUpzfWMsYq1d7cftftUaoefXmUqBT05btzBbn+2XupHhPV8zs2+QYtvZgQjcIYuFYEkbyVrV7tEqTVjXrpC4hWB3z0ke/iyjm3xmGxEzVaO1QRaAwXPbG6+qq2RP+q+5/I5yKvGupG7L7E5NpB7OIfyNs0aYvo7MQ0epQB96fuDOWavzjhOMWCaqjB7IlAkGLc02/vX8WKtRj89cDRR6M93XK9MPQCyTUmxRMmFRAN37vdmugjzbhnGrcN+T19Ht8mkMmOzaPFuMs8BrvwtjuYg0Qg1+/oRGV4VjuzKMDDYEMPEgZxI2yYqf4zEYUU8ybPqHsHk6/oGyTqz/lsMKyo4Iw/qyGrYqWpbpsaiymDxqyPNaEKuQFH1RjgVWlQ5PHKq0vhcsLYYiyy6AY6cBm5Pmme8IYrsJe0KTC4b9ZWt5W2o/x0Q7Kj3jQSaTTQxtbT1U4bLaAj/jnQDZVAnPRM12nnrwdMuvO4AmddGwS8NN13A7sSTcb9AqCRoiAo1n/cIL7U0Xq5sXH38dsVA6oNlqZ3+JkdSkOKlAsuuI3MDt4VnrjR9fFj/OJVqnR6LuSLNZzFP/UMrZBFbSaMhzqQ9Cl3PwRXI9YONvCtkFfH25V6ntCq6dPqZTaNWdEkz8Yy8KB40PxU0kMvbf2Xw0W/RfMr1J1HD6oyIamKEt5nJFbxGKCzZnQ+oCDPqMyfMt1FmO8HcJEGnKJbS0mXmYttUgEKLp5Xi19mXKcTMVFlK/OIdQH4LJ6O8ETKfpTmAHl/0gf8PVZiHAF+4i04XflGCj0XD6YTBwmHxnUfRlmnqs5Hl/ZNHBjOIY+XMlgFT5p1ZcPz9wFCyV7T8wXaVWrnFnTILadBjvLLg8/UxmBz91IHzSGrpVxtJz4pzMUgBWYuHNhtTB/SIdgOlbvcB78I1cxLszTqLqfDYqCQCc7TqM9aPm+hOrlKRjOgNsOadGhtc/Lw8F4M1a2lgbF/sGqPSF3eZiFfYdGnfnCCKT9fCLg01W3SMAS7/38cWGzpqIaGhvqjrl1M1MoeH8bPDfkAiZXgoV5Sk+WvyVwAGcsQQwtAB6ZrPnj7DrRGoXWzGbXFY0MjJXjbVIDpPB1D51Bv1VUU5P5SohqnWUtPUNVNGIcfVeZbkAjuYuw+uhxDqpqNMnQrfFsBzQSV5o6hrJwtWEfVCwmg7NrLLnuhsl/W1t2J5zBCLq60TUbbBSaLvP/yDYp+SNn3OfsQXhYVdx7w9aCvQ1h6FvMSHiwdEdCXMpkqjeNcffyRsuCiVCQ5P05ML7WNjUTcmWTD/zx7EtCm7ZDMMs+KB46f4bQdKn06GvNOCM4y88gIXQ4ViV7n0Mxt4MWZDhbVxW0L1VGSk27aM3bZTAZFBubT7WeDdLXaBOV6YTkpF0F6iIpVtBMK7MH3NvfD9k+3sxtJlZSx8DlwGyXZCl0s91adZ06JKo23lmaVx7jyNAXgYP5qnZj0n7Uc9hGQFiBU7yWPFJ1DcNOP3xsqx4pTjoWwxnSAhIiyDl7ukzNQ9v6xOK8uLyDXmPojgFEHIfpd8ZjjUffyypKK06TShJ8Jz7WG8iVeoSW6UvsWnUW9lFTQvFaLMTyNM6nqRawFjcBEYoOpgrRB7m7PDtNQRkAhtOQMSdtNz2WSzUu7TRYWfJj16pZNhPs7PY6mrNYwuL8ioCBp0D8tGxdCdTvDIi4xi/LN2hVjR5xqN4b0ENW+UqUQn3zxqaEzkI5jcJFKXuo6xfz+Qb40Gq1Yb0ZnAgIUUgY1kE2HCiGcUw2UjljdEZmsqr0gxI0lEoKbfgdfAwvi0Ak0D6+aUYsRCGhQEh3gl/M3+IaIBlOBZ4TmOv+VaMu1KmkJMIrW4AUChLEZjGcpFIA1nB2IfDd7ZxiCQEwQDfTmqWWrkfk7sbhQQ5G9KBBBk92rUWY0ZftN7hpbLx9LV/dKo3mZh+H9Qq1aBJ85HJ4IvSNGVLq9PwRCc0TJPy3XNtqHk2cp6OP1IaOWrWBZc8QJ7U0tdFfD9oVuumtaRuYn0Xr4Ob+e1HBSVCVIyGdXyCOApu1QW8rQZ2N5m3k1bT2Lk5zIVRKeWkzrVB4tLJ/P4TuGD3LXC4Fkq3Ttu4UAlUYGdlyfz+wNl7AThFgLyI4DYcGSXBU1fyzJmgUYm6wQdEkSUKvvpf03n+kqeynEmbwpZvZkPCD3Jnp1SeAN/+QXhfZgq4LwuUE7EEvdnZwmkSnt6HHnKVcXq4OBfk1DrfWI/R8trW0JHG2cbtnBK6TZoBqpbyI4oVGqXZOh8g37uwb8NpIZwj9QRNPdGwm4mTBUjf7EfmHiiD/BHmRJeIUz0Xvpaf0cJKl3pjFU0y7bdYvSU9GkBOM6hYCNwmFuLHCxZ44NqVzcjEqV/6R9Mhf1oFAsIKymp3wvuG7aHYXFgRTq6p0ulsbGu06/vBTT8bNLAVHpRuyEOPaiVkmtxVFDnvjvI6xhSx41Er+nM1fJELrbJehUTrYfxSETdhH9mnHQqjT7fuwPFRM+5qSqIo2RtcCYv24ILIZWqPbUU5dWXhdgt6OF8j+Agvk/tq9MjB6RdJzyHvd3e2dl66Y4qHCZGSBP8H71H+Pf+s5tRQJ+Qp/eGqsCwi6F3PnpGRYqaMJ6Kqs8ry6xtHPCrPNcSW3wZBw7TdL8fl2T2+m3tE+xj9yGe7b0pZ2cbVu1YM4GgbzNpXIbAoJIBHwCnYH+bdKhq4X8rJ+IWI1upv4CD0PkJYT1oRFM9Bu5dtPYBT4t79Kd4Qkp3lq/hASEn0VgXJQ1iZjakPQBbpZyGLVjk9rwLgNRUgTcf1gjyZjJlxOKwE/+Rrmr5vr9fwtR3lG0xupA/AeIw8tgF8X8TaRVxt3WIWyLsCljgtmVYCKPxL2SJK9mjU44k/qvwrGW7QW0ju4l1jLh8yHlDQLL92hJt6EIEHR1X5DdUKZzEpEr1K4hsF7X4Scx0bHyu7bnA5gyZ3A7l/iMbI1RFXTA6DmcxE8CdWh/3lCUsDEfhD0F0uMJwHO3jGkRdTnLJFbqFL4BmLILq/HIBo1PXgzgNn5sFHnTJ3DBYjmbVaM2sfS9FQnP+bdJLKbGrC8xOWBRr5vbNrqpQSObvi3Y8bIz8OEQCBbdEcPE3l7+1gZ8/b8F2DUZR5D9+o+r3MIpN2pm1JknO1t2m/RFW3vnqLuqi+wS5Gs8DeSzLc3VdnXqsyZbFzhxi5SwiStxrkGNmy2VnIn+QImqwCmCeahJwdd9lTsWpKSVXq4NLcJu6yCAjazIdCqXSDIz0eYV9J4Ag6tFjmtbbSGmUZZzJHGt2l+1V7gsmnGVUF6yK0QtOi00zNQnUW/IWzsKBDqv8YSRW+/kJr/Fqe6CxB5fZ7+wkh4+5vQ9KfwnWNBFmFmJ03zNImUudTqJ9EYWgc4T6T3p6WU1MXm3lYIkUa8NqAKZO2tmyn6xI0WhV3Bj5NoOqbNnVRQqmpQvfKg7a7LEzN0Du8vOOTI+vmL8R81YrLfpEVqqK7vAwlFqeqrGAHsubeh2qbb5R6oEbkhkbHmJyzsB8M9E8yn5nHBB4CeAStBhd1XCwDkF2zWuumsmgQrcgeCGUmbB4G/1UymwoCMpmh9UavIeyEzjw4MZ9OzDvQlVOxXqIeE+N1XEudRd1YIu/eLeDG6yvwCRt3RSuXKqZYVUIs5KcBk5/uD+NHZr6zYsP73vpQ5x0B6bP/2wppS96l+FH3lRDxgf7d3EDEflxB4+mTMyLrxeP2vNu0PlPnVBr9Pf0jXHC3OGxRN3GYIjkIpZlLGCyZCyTWVaKvs4mY3rg1h0cr1r0nkB8QpcMDmPLQm+tLlQnO7nTA2AuD3BG6XJNWytK3DTvWm8IGRPtIzaBMU7/1QmnNxaNz8zsR5akLXGTk4aFuSDXcJAT/u5/fqSKcuQSmmeqFH9CoVX73g89FjJC2ZewwNqv6W8h8hN5n5CfL8D2nYjz7cGUp6uiCWkJm8/YIGeyVvxLqKgz8TezRzK4O7ceU2fCsZTu7m6ytUiX6O4HGbSmhKzJv55m82z19GFmIkxZK3v8ihbXR6Jwlq05W8FeC0rnBHGcvWJAEAbOEH7hHYmeH2TmTDn5HYyMLPuF5zuAsXMaLZwBKbyTWS5t+Z0CkzENsUETG/rP4DpkKAD7vkpu/1VFD4BwSzxr9ZSefqkdvP30gG8JZ9tUXaw8SzQgwyVESdJi5xUuK+z4SD2eZGgfG5Cxgs2WJNU59w0qP6TFa0Tb85BqajWTEEOz1WKgkajTrj5hfmtrIf5rFpjEl+fr8ZNdUPVAQVRnDzSJtppDQZALhLzPY9HLpwDmDkGWW9sN1uJvrlnYYDpT4BrQ10qZP5Vl5MNwmaS39iqJ0Y57jiP/7GtkRVeF+9s4R3oA8lFZXP6bxyKT86rNIc7NcRpfHFDYOrD5tPQTp8hWqJnuTCEvFBxUe6S1EeK+tvYMwdapRu5KCLLahATRCLLzLS4+p6bDBJJGvKhYrljm+cSb3D3ZhMBDYYaBTShXMmGjQqZ9sUuyo53nZpgtv/oQ74/V+D9GpS4Ol/78Xcj0b3CHkrv61Fhul3vqhy43QbZFMotTf7hpyGQM+ZP9JeBx6vRhbXi58CTEFNNlQhF3TUgJ6yq1sctfXq+vlnCzLt7GN80bCMHFnTXW57c10GQDJJMGdjPQGRs8F+5klADdOHJPb8dOwzCpmmRcGvmDNj2Y5WepRIEaUpOchKvSG+j/uqKwPYEZkpvABDJUDzXVuGjtnydUK7S1pdUnx/TrQFRVUQmQJvcu3UKnHnuNxc8RNqwwjbcJvGVtsJdJtMGWTUrN3eIJSgCgu4ndeYZ9gZIjPEb8XwBQhBZ8srEHkKwHIUIx/mhRO5NTjtIgkYYSMm8XXMa79tePDJCR1HULOWt4Hwj/H3hwpdD3DZAgvd0cvdSuezdkMZz35YPCcP3FA+DjVtjNyf0TvUvdYdmZpvZ66yDzP65gUv2NqXTtppOLbc60NuuzBEAfSqydhHGrjyF7vDmJGeiTYMTP6SBxwp8pPLpAKEKsJFjg2FRlwwG+eyYxzB5bvfmbe+uXrkTtzXW9G9Vc3L8Hi7MVn8x1IqXEZRuT1HBVf1xo7h6bnzsWkqzlYjZqx0Q8HZg2q+Vyy1w6wrSbYt49JRC8NUx7I6+pEqojD0ET9VhKD8fRY1unZTDZLnPW95VZ5Dfg4kT+CA9Bo7PX7x8g/Etga0xjJ3lB6pbyjQhGXT91q9KzZA8oDGuNYlSP9EBMb7oVbb4VFvp8DdiFfzEHqe1wnaN+1bcBJLsojPe+hEfCVhwFUb0uNtP0J8IHHXn+HxCZ/TkmKMxV9mXlu+FHhQUW0fwTOuqS8C4MtbGEWHbDc3xlyuwg6GKhDNDyOaXuMWvm/qzqfUZVm1TsJbvzpbgxK21Kji1K9N3RQ5yJ6XeB8J2Hq3yWE9vjllFUGjjMe9B5jXVbYdm6Ce03eiEr1cMNjWbCdgb7iI8HW3hvrXzMMNR0DB7wzWHKLywCN3nGqh5Ph3XdJJIr6etRN4B0UUnfaq+zzAPc4Z8TZn80cvIG3FmPiyJTiR/tmuUC2GnXh96S/XxVXhqZ3bfj32EmJM6KSVkjwULf6hbNa4YZnmoHpRS3hcr7w7HAZNCwoyjGEiPmJNCszuiY/HPzs+b3rKCYX1Kzyb9lfqGnL5jR8SS1EX7XB4hCBMdSb7l0gs4aR2Iyh5Dp4JkUiRiswnqEqVfAY/UKJIRS9/q6VhaypXtvqbXgUwMoktTzUz4sUyR9gK10i3j3APuPkzJXGfe4cPI4A5BiukHFVhc8zXoO7ZFjyFvP9gCrADZaQ8KUaK7n+hG3UmBiwiNFmx+YIZPwAMNHv/e2IJh1rDh7ZZsEIR6b+1rD4Y8Pd4bKdp8vQ3wgqVp+RhVWhRq6+vSBoWI6oVFWaLSV5+QrGrEJMrcXo7UIyXQTqSi0ji3OkCRTtMB6sjOj4CLbHUoO2OCOGMJrXGcYL4YJNBfxZiV6BNcpE9NLe0HLWS42IjZCEjyfIs7H0u9xL6WcWSEtPK3XdM8yGlWWJLqt3Y6JPKxAN4zsXeZ1qLPAKfDLN0rMOVfuUbMPGOF5EMs7zNzGO9Lw1DBmvEs1D0N2lNDMJGyEG3ZOxDh1ZhQcV5Si9RmjKFaeRS6f92i9PMxg/TQeV4X4aThGU3uyD4/jQe/RmwZ7U75A9vFBuAJjN7j6e21kd6kjpJ5IDrv66cjVlFJQfVTZQGLMfeHbb3tERxx/u1HQLwN+sOnuwsphxZCEiSAVmQ0CVrTUh6NqW30HNP5TYc8eSmJXMsE3dl0dtz8WqrI5eQZOpCJKbCJgx1o3mwKGIvQAAANlVI7fqzbyRcPrEKpUvwavJM3r3T3edqmr21uwg9zVu2ezPdHwr5hAmKHL3nsFzDUbLIXJfAqSJEVJOe6bK9QsQ9EaNV7yXwxrPFVI6zd4JXyXYkIh3t6URSivmC+bQEgEwBSTyjB1Fb7PHtKCAP2yu8UEW0p6MJOt6nAZ7mipEtapVVqvYFvzc31jt8gBIaRCIpUM51Mh4qFaUlzyGi6Mea1fw3gPWEoqqTTX3Qwb0LoO2cwL2dA5G8oltvE9GhpgmEeZJbxWZ63mZFvnQ0vK10AwdmqpBqyxD08jO5D2ChbS9QpGSULqYjrFLyHr2hi59fiEE2LQHNfJyLMpgyY2mw1J7fz

nSUlJqBSxPg+kpaXBYrFg2LBhuOeeezBjxgyP6y5evBiJiYltj/T0dFna70IoJfIQnjG
ZJKvyzBnXz8V5kJgojUU0Iu2FkixK40BFQ2RH82QeUzs3JWPMZVl7tmzZgh07duDll1/
GsmXL8Pbbb3tcd968eairq2t7HD16VJZ2uxAKVXkI3/CW0BMKbleALEojQ0VDZEezZJ7
u3bvDbDa7WI/V1dUuVmZ7Mn+5CV1yySWoqqrCwoULceutt7pd12KxwCJmpVeSUMh0JHz
Dm1CGQiIP4JqQRhalcRBFQ2bO5JYkFQ0JGs0syujoaOTk5KC0tNRpeWlpKUaOHOnzfhh
jaGpqkrt5/kMWZfhAFiWhd6hoiKxoOjxk7ty5mDJlCoYNG4bc3FysXLkS5eXlmDVrFgD
uNj1+/DjeeOMNAMBLL72EXr16YeDAgQD4uMrnnnsO9913n2a/AQCvXCIKRxvdkiA6xhe
hNPp50KULj8cyxif9bS+chP6hoiGyoalQFhYWora2FosWLys7Oxfv16ZPxilVV
UVKBcjE8EYLfbMW/ePBw+fBiRkZHo27cvlixZgpkzZZ2r1EzjHjvHMMotFmvSYCF3CwfU
aGcl/56lT3JqkIU9EGKN5CbvZs2dj9uzZbj9bvXq10/v77rtPe+vRHY5u1/ZBdCL0CAf
XK8Azd0+dogIaRNhDd3U5oKEh4YUnoayv58ICGP9cKCmRrOOdO2nAOhHWkFDKASXyhBe
ehFJ0mLp2BRISVG2SrBw7BrQvCUkD1okwhoRSDqgqT3jRkVAa3e168CCPuTtCA9aJMIa
EUg7I9RpeiIo7noTS6B0mGrBOEE6QUMoBWZThhSeLMlQyXsWAdbOZv6cB60SYo3nWq+G
x2aQptsiiDA88TbUVKq5XgA9QnzSJu1v79SORJMIaEspgqagAWlv5uDOqXhIeCKFsaQE
aG/kcpEDoWJQCGrBOEADI9Ro8wopIT5dcVURo06mT9F87ul9DJUZJEIQTJJTBQkNDwg/
HqbaEUJ45wyfJBUgoCSLEIKEMFkrkCU/aC6Vwu/boIbliCYIICUgog4WGhoQn7YUylBJ
5CIJwgoQyWMiiDE88CSWdBwQRcpBQBgtZlOGJJ9crCSVBhBwklMHAGI2hDFeEUJ45w5/
J9UoQIQsJZTBUVwMXLvByByXtBhA0klMEgbo6pqXwWeCJ8aF+dh1yvBBBGykFA
GAyXyhC+OQnnqFHD2LH9PLniCCDlIKIKjrNEgmSCPbGc9GJJ9crCSVBhBwklMHAGI2hDFeEUJ45w5/J9UoQIQsJZTBUVwMXLvByByXtBhA0klMEgbo6pqXwWeCJ8aF+dh1yvBBBGykFA
GAyXyhC+OQnnqFHD2LH9PLniCCDlIKIOBEnnCF0ehFOdBcjIQE6NZkwiCUAYSymAgi
8cRRKcrsSREhDQhkMZFGGL+4sSsp4JYiQhIQyGKjOa/hCFiVBhA0klMHAA0klIHCGLlewxkhlE1
NwL59/DVZlAQRkpBQBopjpmOvXtq2hVCf+Hhpqq2yMv5MHSaCCElIKANFWJPyYDVqm1
bCPUxmYDOnflrUZ2HhJIgQhISykChRB5CCCXAhZM8CwQRkpBQBgol8hAiZ0M8CwQRkpBQBgol8hAiTgnw6kwWi3Z
tIQhCMTQXyuLiYmRmZsJqtSInJwdbt27tdDst27ZvJ1cxbGLjHwDPDttSwbcmzxu066deswceJE9OjRAwkJCSjJ0DmJCCjNzcUnn3yiYms
doEQewlEo6TwgiJBFU6Fcs2YN5syZg/nz52P7du04AMBo6NxQQ0OpiW7R+amCyzj55sqx
r12Pnzp248sorce2116JMJFOoich0TEhQ/7sJfeAolJTxShAhi4kxxrT68uHDh2Po0KFF
Yvnx527JBgwbh+uuvx+LFi33ax8UwkI88cQTPq1fX1+PxMRE1NXVISFQkSspAWb
M4K9NJuCVV4CiosD2RRAhx4KioskHDFBXgmB3wjnSdCsLFiCOzGNiQzUE6OWyUo3cDfTYXjuTanx7u6ZbHaYJJCzXiQrrpkKgsKHYzJvfg1
aP2+ohpGD0o36+C3Rq8NFtUD6fwSyV5RbWzWA3Tw/OZTlGjSaSq2Cd1vYg+4g48EzL9/H6LKn9XWiQMSh99Yc2fM6MPqDFh1Aq
o0Ak00ZQ8bxyePENQmqMmebTffAV5l2zi5zuC4qGhmBJZvBGhxL0SrQMhdrfDeGuvL6vLbe02r9sCChvuHPr7+kQj+Y63C+0/GuoacVbWPrQyMcRwOg+cW+EPfA1HPy+M2uDA1qfOz9DGtLXZFT5Wr7IUrlVm5y3OUsn
RPyj5uo8O1lQJ9KQ1lyB9T2rt3jvExWPFiNe5wMDMrwDSxTDodJxuuv+xRSfb3xYdfCVmkE7XhKCgKCjWbKjrGsgmIivUuRDdhqlv2Q6qzuaovwXs+y9Y6tN/TZ5T0ynb8XpF2GIiQ7uygtyqpbL1aNBbqWZkMvF2Xby3Dmlc5w3GhHxpZJ0hiy6dApMRLH9gD7PiT3JZb9UDChgDsbTQxZN/ho3Sfn88SmiEM1gPuJr3WnjQfmrMQdFFwtUK2ybFMe4x+PzlfP5X+T2cKJBqqaaedgkjbOQgrQ4c6skMqF2Y3Axg44GNdq82ciPpOuIXvuwpyI2DKTqVU9hbcP8rcjDFkZCtHEbyEwo+bmjoS4FrfWCM2mfeUKHMJzY9mTPm0jSWE4yWnQ4p5+7H8B4Bo9vwUuUJ4tXY2qGmD5v0YL3xGmHKI1fhCxrl9ngC/lhqHK3XrAlCeIlyf/OYtzvSUGLCLcz9PUoAyzzYiQUgfXqz4qqpLfBBJzWCEOVQfkRTQEcgbeuS+QRfGNd6uUDEGhsPXtqfOGGXM4Zhy32LCLl4UMyxcc/VZCK5tVvSMf8nx7vvC+Y7fVfq8T7Bj2Z0MKLlJ37H3WrV2RdyMEZ6oGdXpmUtZs/J9e+9o4yqFo1nbCfChaPPm7gd4Dck2Jsfsoh6AQthe4YXEnlb8EUg2vTZ1hy7yUMUyi7FPvNcQNsHbg8ZcYVuIZ7GRqBC2XqUWiZlWtOYRCVQ8K0VbQcKyG1Tgv3P6jElgimSkQ0jR7Aqvv4RQhCRB2H1GXp1j7+dXZcSd2q9ykzdDDMVDJTIDQtrZg+kKn/NXKEcI6r0W1YSWiZWFHyUK95ghzS7x9NHIrFaNTIYSq9Wb1QBGpnNMGF1vG6AxvwUYAU1H2Pm6FLpl1VpQJnPLeIDeQuBmkzahuT8Wtf8nVn0iS+3ZPBP23PcJoRuKJfyqQrMTKC0S7hLGCKtvpYH9xKCXfbjeGsbT/TIQaEk6LjqPa7N1P/Xfm9jSN0eS5zqbe3x4e1d4K0VBbt5y0sFD8P4l2aDHqU63Flb6yo8CESeAz5bzHbaWjeNqvGmHz0dz2M6H8m7E1aqAAzfDT1IY8lOu3iXw3MkKFT7/TdJuNFDx0DD/u/u91HOjvYEwb6ALoP6GoiN1ZD6/Q3x3Ypg2Cd3aQh4aN6BKT0TiFCEUXqcZmOG07V6yBrV4qk63tOI3Ki6PjrNUaF4aUpxTGPrGhcOcAyVJGo+1ywqIxsO8SRnSf9z8D1wspSiCBJ7T3F4GTjD+e+EUhYcW/pS4Nij3EmDi3+fhsDhSLOq7jA7gP4Cah6Akr2+ML8BQeTtO55s3OAX0wOakWhTKT/h5bWVbqTFbj22gsyKThaCj9TFHRRGPmKG+KvDSaP98b9BlF4mZRUQyEJoMoQ9TcSUNlZsrMIO7QK+oaEGNjjiqsAI5MdnDfQ/bZNV5pkeU31bkfTRxp7UoNzbUcRjiXpSlnq6hb/IYhpH3hg0CmUzLXN3+mdtLoQ5eVfcrN8Z81OdMRuIEV5Sx27jBNTVl73CvzqjvGCdbVxTd5UFKlfZxtNPHkCgBj2IUa5x+cRp3YAFoI/jwQa2nJ6vj0Bm6dt7XlAhq5PBuYqRnsr1O/2eXfsuDwW7lsr6IrBk6lJ0FpVuUp2zO9F4Qv8D0K3Y8uNR5EbMjDsl2/xn2v1P1MFt0y1BaQkCM

qt/oYte3Ubn0Sdsk8BEEQBOEPZFESBEEQhBdIKAmCIAjCCySUBEEQBOEFEkqCIAiC8AI
JpReKi4uRmZkJq9WKnJwcbNmyxev6mzZtQk5ODqxWK/r06YOXX35ZpZZyFi9ejMsvvxz
x8fHo2bMnrr/+euzfv9/rNhs3boTJZHJ5/PDDDyq1mrNw4UKXNiQnJ3vdRuvjDQC9e/d
2e/zuuecet+trdbw3b96Ma6+9FqmpqTCZTHjvvfecPmeMYeHChUhNTUVMTAzGjRuHPXv
2dLjftWvXYvDgwbBYLBg8eDDeffddVdve0tKCRx99FJdccgni4uKQmpqKqVOOn4sSJE17
3uXr1arf/w4ULF1RpNwBMnz7d5ftHjBjR4X47k8mEZ5991uM+1TjmeYA3B47k8mEZ5991uM+1TjmSkFC6YE
1a9Zgzpw5mD9/PsrKyjBmzBgUFBSgvLzc7fqHDx/G1Vdfj
TFjxqCsrAx/+MMfcP/992P
t2rWqtXnTpk245578OWXX6K0tBStra3Iz8/HuXPnOtx2+//ThQ4e/dQH3PFYIRbrrjiCjZr1iynZQMHDmS
PPfaY2/UfeeQRNnDgQKdlM2fOZCNGjBCsjR1RXV3NALDQt9MzZ9XGfDhg0MADt9+rR6DXP
DggUL2GWXXebz+no83owx9+vfrvF6ZwPDc2PXKTdOZAFkmqcsimwYVbcSHaiTXQSRgx39NJqyYGXLDZ9hvO1mrdDZdrv8hCVKmgjjUAkCXvK8nHX7CExbgmTq7o5Ij2jdhsvBKrBKb7wHOZn8N5HvHZCttWQQtX8XgKVSsmfGrHFPjfGgJtHOuRQwUMGF9sjUyl1exEkLrdYUhKlDyf1zrSLBHPOGCcHkRcrbKVuqSYdxBFQuYnyZUAB/ypfqSYSsGTbzg7RJwZ/nUp0ZJIWTxQyFSnfqBwT7H9BsvRBqdTPvMWdK4xHlFzHpFExGQPWHyVPG1y8Ap3uHJOX5XKqiFjUBPsOJTivuLpZSiVkzdTgkfgqbkEdWZl5BNBNnwF1jgR1k0tb4MKpk5HpE6bR5MVLqdOW8txXE7GSHE8/6vYMEMsMdyWkYb6i4S8RqWHJKYCPuMpY0njLfvnwLPEo6jnhCnNQIdfwXAJ8kaujnSPfFlF8CHXymCKUf6D76F6VAjOE1VV3PeiSRu0fz77oQbcQKfnP44xy1zjOhGEHG9uFBn5onzKaIjjrB2PFQDOANtTEgMSa2yVXdOEcZjn3m2WGE4PjdrFWZ5tRW5tq4w7nUe0AFvJTNyhGCdDUJZmNS7y2AEfWRx3AmCdqHBYfV0t+UNcBpCUz2C66lkD2rbfvIOXFuARQHUjHv0N0K6mbzNUrTK6bzV1jW3LhNqkYGXxBz39dfYTHgA0xfmXvWfaaxi4vefx3Gn3C4dwXMFiSkYDORF0xsryY/92ZBmFw0f4HTq8PvuhEe+8CYh3P01xeaeepPoe8HyYWhHuD/eMZTjn3H8Ft/VcYbKPN6pwJT8v7bb5F7Lid5lM1Mwf4TuPC9f7JVWlcVU3sFdhhfu2b3bwfnMvbEhWd8ZfAjH9e8KDqMJ5G9jOJm8X2C3mbCXrdwqzgcmySnwwkB7gjb4Md1WKGEZYQ3D2qwgKRuaLCVNGzg5SvkGcu4y8TqbE9KvnqYD2r6MBpmGT2GfjBGC3wAAq/z4kWHG6WYZJXVdhy8YrN9Bk7BLmGt9l79jMkW7eYRkbdsnrsdHxUyjvG2HKc/kcsD9+j4aYkdX2SxmfIwN+GDI5Mr4dTfD/Gq0C3dOmN+RgcLXedMW4zlMHRn/+DtVLvP8FJOvPkOQ8hAnCDqZAp8G7mRPcDmMxmeBjEbrkEepnU/a1BTKJ1c8hFjDr7R8yOOyOEkyeqaEf7S+zyFvSbq4/rAgwSpcYB/wHQW8BX8sM6cu6lmrh9YpZMeG+3bvMe6nQ5nsmtVo1Uw8Atu1nvkQzf/8K6fZr7XYwv6ss4FxN6/2fHnxSBeuvuhqN0k4TU6/Z9/mfiSuNEFXK2f6Mh4OJ/Qfgu4c0fnbt8SbAMGlMmumk3ANmI5t2qwi3Ea3cF5GRZ/V0ONw2UhzLmvz8TWl4OW+1lHdBNPIb6+rZUHc2OXDAGqXh1DlENG4g8t0BE8vbxPFubZOEFzpcFwEaFsOFUbJ3IYe+qmxF9g8SKEhkGlWrPYdK8ZvTPmIbo6EBX49S5GRZfQXWLKmljVWIM0QNVNWPbdBpJ//YGdNjLiypf7lOB+G/u7f0lhUDFqWVNGEVwFxGjv9X7I7fTw+11bFA/IlNHbkv6SqmP5CzoeOY1JZdTV8bqC80KQFZfg9

NY3l5eU7rb9y4kQ0ZMoRFR0ez3r17ezyBlAKA28drr73msd1PP/0069u3L7NaraxLly5
s9OjR7MMPP1S13YwxVlhYyFJSUlhUVBRLTU1lN9xwA9uzZ4/HdjOm/fEWfPLJJwwA279
/v8tnejneYlhK+8e0adMYY3yIyIIFC1hycjKzWCxs7Nix7Pvvv3faR15eXtv6gnfeeYd
lZWWxqKgoNnDgQEUE31vbDx8+7PG837Bhg8e2z5kzh/Xq1YtFR0ezHj16sPz8fLZt2zb
V2t3Y2Mjy8/NZjx49WFRUFOvVqxebNm0aKy8vd9qHHo+5YMWKFSwmJoadOXPG7T60OZ
KQdNsEQRBEIQXKEZJEARBEF4goSQIgiAIL5BQEgRBEIQXSCgJgiAIwgsklARBEAThBRJ
KgiAIgvACCSVBEARBeIGEkiAIgiC8QEJJEARBEF4goSQIg1NdXY2ZM2eiV69esFgsSE5
OxqRJk/DFF18A4NMyvffee9o2kiAMDM1HSRAG58Ybb0RLSwtef/19/119OnTB1VVVfjPf/6
DU6dOad00gggJqNYrQRiYM2fOEuLti4cPOoEuXLti4cSPy8cPy8vJcPu/duzd++/vnvntvcZGRk4cuQIAOC
DDz7AwoULsWfPHqSmpmLatGmYP38+Ic7ciOTkZDzzzDO46aa
bVPltBKEXyPVKEAamU6dO6NSpE9577z8z00NTW5fL59+3YAfL7MioqKtvefBr/9dtx
///3Yu3cvVqxYgdWrV+PPf/6z0/aP/44brzRnz77be4/fbbceutt2Lfvn3K/zCC0BF
kURKEwVm7di3uuuusunD9/HkOHVeXh5uueUWXHrppQC4Zfjjuu+/i+uuvb9tm7NixKCg
owLx589qWvmfnmm3jkkUdw4sSJtu1mzZqF5cuXt6t60zFMXFxer8OILQAWRREoT
BufHGG3HixAm8/77mDRpEjZu3IixAm8//77mDRpEjZu3IihQ4di9erVHrfZuXMnFi1a1a1GaRdurUCXfddRcqKir
Q2NjYtl5ubq7Tdrm5uWRREmEHJfMQRAhgtVoxcceJETJw4EU888888QRmzJiBBBQsWYPr06W7
Xt9vtePLJhTE0mCMNAFiVBhCCDBw/GuXPnAABRUVGw2WxOnw8OhT
79+9Hv379XB4RExdJt4csvv3Ta7ssvgvgQOV/wEEoSPIoiQIA1NbW4ubbbcMcc
99FLEx8djx44deOaZZ3DddcB4Jmv//nPfzBz09HTcdNN
NiIiIwHfffYfvv/8eTz31VNv+33nnHQwbNgyjR4/GW2+9ha+//nHoh1JSVa/VyC0AZGEIR
huXDhAnvsscfY0KFDWWJiIouNjWVZWnsj3/8I2tsbGSMMfb++zfv36scjISJaRkdG
27ccff8xGjhzJYmJiWEJCAnriiiivYypUr2z4HwF566SU2ceJEZrFYWEZGBnv77bfV/ok
EoTmU9UoQhFvcZcsSRDhCMUqCIAiC8AIJJUEQBEF4gZJ5CIjwC0VlCIJDFiVBEARBeIG
EkiAIgiC8QEJJEARBEF4goSQIgiAIL5BQEgRBEIQXSCgJgiAIwgsklARBEARBRJKgiA
IgvDC/wfMA8JN0DJwqgAAAABJRU5ErkJggg==",
"text/plain": [
"<Figure size 500x400 with 1 Axes>"
]
},
"metadata": {},
"output_type": "display_data"
}
],
"source": [
"logistic_map_sonify(r=3.8, x0=0.2, steps=20, alpha=12.0, base_freq=220.0, note_dur=0.2)"
]

```
  },
  {
   "cell_type": "markdown",
   "id": "2c292e46",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## 3) Future Direction Snippet: AI or VR Concept\n",
    "We won't fully implement VR or AI here, but let's show how you *might* generate an AI-based chord progression for certain data clusters, as a minimal demonstration."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "id": "bdfab08f",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
```

"text": [
"fake_ai_chord_mapping ready.\n"
]
}
],
"source": [
"def fake_ai_chord_mapping(data_points, cluster_labels):\n",
"   \"\"\"\n",
"   Pretend we have an ML model that clusters data into groups.\n",
"   For each cluster, we define a chord or scale.\n",
"   We'll just do something silly: cluster=0 -> C major chord, cluster=1-> F minor chord, etc.\n",
"   Then we play short chord stabs in sequence.\n",
"   \"\"\"\n",
"   samplerate=44100\n",
"   note_dur = 0.4\n",
"\n",
"   chord_map = {\n",
"   0: [0,4,7], # C major triad (C-E-G in semitones from root)\n",
"   1: [0,3,7], # C minor triad\n",
"   2: [0,5,9], # maybe a sus chord or something\n",
"   }\n",
"   base_freq = 220.0\n",
"\n",
"   # ensure cluster_labels match data_points in length.\n",
"   for i, val in enumerate(data_points):\n",
"   c = cluster_labels[i]\n",
"   chord_intervals = chord_map.get(c, [0,4,7])\n",

```
    "        # let's pick a root offset from val in semitones\n",
    "        root_offset = int(val*5)  # scale up data a bit\n",
    "        wavesum = 0\n",
    "        t = np.linspace(0, note_dur, int(samplerate*note_dur),
endpoint=False)\n",
    "        for interval in chord_intervals:\n",
    "            semitone = root_offset + interval\n",
    "            freq = base_freq*(2**(semitone/12))\n",
    "            wavesum += 0.2*np.sin(2*np.pi*freq*t)\n",
    "        sd.play(wavesum, samplerate=samplerate)\n",
    "        sd.wait()\n",
    "\n",
    "    print(\"Fake AI chord mapping done.\")\n",
    "\n",
    "print(\"fake_ai_chord_mapping ready.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "27feab76",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "### Minimal Demo\n",
    "We'll craft some random data points from 0..5, a random set of
cluster labels 0..2, and then hear the chord stabs in sequence."
```

```
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "ab4f2db4",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Data: [0.98657701 4.75417308 0.2462529  3.33251018 1.40053482 1.01432601\n",
      " 0.93580328 1.3543541  2.36947326 4.83969175]\n",
      "Clusters: [1 1 1 0 2 0 0 0 1 0]\n"
     ]
    }
   ],
   "source": [
    "data_points = np.random.uniform(0,5,10)\n",
    "cluster_labels = np.random.randint(0,3,size=10)\n",
    "print(\"Data:\", data_points)\n",
    "print(\"Clusters:\", cluster_labels)\n",
    "fake_ai_chord_mapping(data_points, cluster_labels)"
```

```
   ]
  },
  {
   "cell_type": "markdown",
   "id": "da683883",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "In a real system, these clusters might come from e.g. PCA + K-means on a high-dimensional dataset, and we'd pick chord sets accordingly, or generate them with an AI composition model. The final code would be far more sophisticated, but this snippet shows how we can embed a *fake ML-driven mapping* into our sonification pipeline.\n",
    "\n",
    "## Conclusion\n",
    "This Chapter 9 notebook underscores how *case studies* (brachistochrone, logistic map) can reveal the power of Helical Sonification or multi-axis mapping, and how *future expansions* (AI chord decisions, VR) might further revolutionize how we experience data.\n"
   ]
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Python 3 (ipykernel)",
   "language": "python",
   "name": "python3"
```

```
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.21"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

chapter9-version2-notebook.ipynb

```
#%% md
# Chapter 9: Case Studies Notebook

This notebook demonstrates two short examples:
1. **Cycloid (Brachistochrone) Example** – parametric function mapping.
2. **Stock/Financial Data** – chunking and multi-voice approach.

We'll reuse the same basic pitch/timbre code from earlier chapters, with a few new twists to show
practical usage.
#%%
import numpy as np
import sounddevice as sd
import ipywidgets as widgets
import matplotlib.pyplot as plt

print("Chapter 9: Case Studies Notebook imports loaded.")
#%% md
## Common Utility: Snap-to-Scale, chunk_data, etc.
#%%
def snap_to_scale(val, scale=[0,2,4,5,7,9,11], semitones=12):
    octave = int(val // semitones)
    remainder = val - octave*semitones
    best_note = None
    best_diff = 9999
    for s in scale:
        diff = abs(s - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = s
    return octave*semitones + best_note

def semitone_to_freq(base_freq, semitone_offset):
    return base_freq * (2.0 ** (semitone_offset / 12.0))

def chunk_data(data, chunk_size=10, method='mean'):
    out = []
    length = len(data)
    for i in range(0, length, chunk_size):
        block = data[i:i+chunk_size]
        if method=='mean':
            val = np.mean(block)
        elif method=='max':
            val = np.max(block)
        else:
            val = np.mean(block)
        out.append(val)
    return out

def play_sequence(data_array, alpha=4.0, base_freq=220.0, note_dur=0.3, scale=[0,2,4,5,7,9,11]):
    """Plays data_array in a single voice, snapping each data value to pitch."""
    samplerate = 44100
    for val in data_array:
        raw_sem = alpha*val
        snapped = snap_to_scale(raw_sem, scale=scale)
        freq = semitone_to_freq(base_freq, snapped)
        wave = 0.3 *
np.sin(2.0*np.pi*freq*np.linspace(0,note_dur,int(samplerate*note_dur),endpoint=False))
        sd.play(wave, samplerate=samplerate)
        sd.wait()
    print("Single-voice playback done.")

print("Utility functions ready.")
#%% md
## 1) Cycloid (Brachistochrone) Example

We parametrize a cycloid, then map `(x(t), y(t))` to a short pitch sequence. We'll do something
simplistic:
- Let `x(t)` increments define the time steps.
- Map `y(t)` to pitch.
- Possibly chunk so we don't do a million steps.

Cycloid param:
\begin{align}
x(t) = R\,(t - \sin t), \quad y(t) = R\,(1 - \cos t), \quad t\in [0,2\pi]\end{align}
where `R` is some radius.
#%%
```

```
def cycloid_data(num_points=40, R=1.0, tmax=2.0*np.pi):
    t_vals = np.linspace(0, tmax, num_points)
    x_vals = R*(t_vals - np.sin(t_vals))
    y_vals = R*(1.0 - np.cos(t_vals))
    return x_vals, y_vals

def play_cycloid(num_points=40, R=1.0, alpha=4.0, base_freq=220.0, note_dur=0.3):
    x, y = cycloid_data(num_points, R=R)
    # Just map y to pitch, ignoring x except as indexing.
    play_sequence(y, alpha=alpha, base_freq=base_freq, note_dur=note_dur)

print("Cycloid functions ready.")
#%% md
### Quick Demo
We'll do a 40-point cycloid, R=1.0, and map its y-values to pitch.
#%%
play_cycloid(num_points=40, R=1.0, alpha=5.0, base_freq=220.0, note_dur=0.25)
#%% md
## 2) Simple Stock/Financial Data Example
We'll randomly simulate some daily closing prices. Then chunk them into weekly means, and map them
to a pitch line.

*(In reality, you'd load real CSV data. But we'll do a quick random approach for demonstration.)*
#%%
def simulate_stock_prices(days=60, start_price=100.0):
    prices = [start_price]
    for i in range(days-1):
        change = np.random.normal(0, 2)  # daily fluctuation
        prices.append(max(0.0, prices[-1] + change))
    return prices

def play_stock_sonification(days=60, chunk_size=5, alpha=4.0, base_freq=220.0, note_dur=0.3):
    data = simulate_stock_prices(days=days)
    chunked = chunk_data(data, chunk_size=chunk_size, method='mean')
    play_sequence(chunked, alpha=alpha, base_freq=base_freq, note_dur=note_dur)

print("Stock simulation & playback ready.")
#%% md
### Quick Demo
We'll do 60 days, chunk by 5 (representing weekly average), and sonify. Variation might produce
interesting pitch contours.
#%%
play_stock_sonification(days=60, chunk_size=5, alpha=5.0, base_freq=220.0, note_dur=0.2)
#%% md
## Interactive Version
We'll let you pick how many days, the chunk size, the pitch scaling, etc. Then re-sonify on each
slider change.
#%%
@widgets.interact(
    days=(30,120,10),
    chunk_size=(1,10,1),
    alpha=(1.0,8.0,1.0),
    note_dur=(0.1,1.0,0.1)
)
def stock_interactive(days=60, chunk_size=5, alpha=4.0, note_dur=0.3):
    data = simulate_stock_prices(days=days)
    chunked = chunk_data(data, chunk_size=chunk_size, method='mean')
    play_sequence(chunked, alpha=alpha, base_freq=220.0, note_dur=note_dur)

    plt.figure(figsize=(7,3))
    plt.plot(data, 'b.-', label='Daily Prices')
    plt.title(f"Simulated {days} days. chunk={chunk_size}, alpha={alpha}, dur={note_dur}")
    plt.xlabel("Day")
    plt.ylabel("Price")
    plt.legend()
    plt.show()

    print("Played interactive stock sonification.")
#%%
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "6b801856",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 9: Case Studies Notebook\n",
    "\n",
    "This notebook demonstrates two short examples:\n",
    "1. **Cycloid (Brachistochrone) Example** – parametric function
mapping.\n",
    "2. **Stock/Financial Data** – chunking and multi-voice
approach.\n",
    "\n",
    "We'll reuse the same basic pitch/timbre code from earlier
chapters, with a few new twists to show practical usage."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "5646e5f2",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
```

```
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Chapter 9: Case Studies Notebook imports loaded.\n"
     ]
    }
   ],
   "source": [
    "import numpy as np\n",
    "import sounddevice as sd\n",
    "import ipywidgets as widgets\n",
    "import matplotlib.pyplot as plt\n",
    "\n",
    "print(\"Chapter 9: Case Studies Notebook imports loaded.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "88b8aafd",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
```

```
    "## Common Utility: Snap-to-Scale, chunk_data, etc."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "1e2d0749",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Utility functions ready.\n"
     ]
    }
   ],
   "source": [
    "def snap_to_scale(val, scale=[0,2,4,5,7,9,11], semitones=12):\n",
    "    octave = int(val // semitones)\n",
    "    remainder = val - octave*semitones\n",
    "    best_note = None\n",
    "    best_diff = 9999\n",
    "    for s in scale:\n",
```

```
"        diff = abs(s - remainder)\n",
"        if diff < best_diff:\n",
"            best_diff = diff\n",
"            best_note = s\n",
"    return octave*semitones + best_note\n",
"\n",
"def semitone_to_freq(base_freq, semitone_offset):\n",
"    return base_freq * (2.0 ** (semitone_offset / 12.0))\n",
"\n",
"def chunk_data(data, chunk_size=10, method='mean'):\n",
"    out = []\n",
"    length = len(data)\n",
"    for i in range(0, length, chunk_size):\n",
"        block = data[i:i+chunk_size]\n",
"        if method=='mean':\n",
"            val = np.mean(block)\n",
"        elif method=='max':\n",
"            val = np.max(block)\n",
"        else:\n",
"            val = np.mean(block)\n",
"        out.append(val)\n",
"    return out\n",
"\n",
"def play_sequence(data_array, alpha=4.0, base_freq=220.0,
note_dur=0.3, scale=[0,2,4,5,7,9,11]):\n",
"    \"\"\"Plays data_array in a single voice, snapping each
data value to pitch.\"\"\"\n",
"    samplerate = 44100\n",
"    for val in data_array:\n",
```

```
    "        raw_sem = alpha*val\n",
    "        snapped = snap_to_scale(raw_sem, scale=scale)\n",
    "        freq = semitone_to_freq(base_freq, snapped)\n",
    "        wave = 0.3 *
np.sin(2.0*np.pi*freq*np.linspace(0,note_dur,int(samplerate*note_dur
),endpoint=False))\n",
    "        sd.play(wave, samplerate=samplerate)\n",
    "        sd.wait()\n",
    "    print(\"Single-voice playback done.\")\n",
    "\n",
    "print(\"Utility functions ready.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "a669ef2e",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## 1) Cycloid (Brachistochrone) Example\n",
    "\n",
    "We parametrize a cycloid, then map `(x(t), y(t))` to a short
pitch sequence. We'll do something simplistic:\n",
    "- Let `x(t)` increments define the time steps.\n",
    "- Map `y(t)` to pitch.\n",
    "- Possibly chunk so we don't do a million steps.\n",
    "\n",
```

```
    "Cycloid param:\n",
    "\\begin{align}\n",
    "x(t) = R\\,(t - \\sin t), \\quad y(t) = R\\,(1 - \\cos t),
\\quad t\\in [0,2\\pi]\\end{align}\n",
    "where `R` is some radius."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "9b3ee7a2",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Cycloid functions ready.\n"
     ]
    }
   ],
   "source": [
    "def cycloid_data(num_points=40, R=1.0, tmax=2.0*np.pi):\n",
    "    t_vals = np.linspace(0, tmax, num_points)\n",
    "    x_vals = R*(t_vals - np.sin(t_vals))\n",
```

```
    "    y_vals = R*(1.0 - np.cos(t_vals))\n",
    "    return x_vals, y_vals\n",
    "\n",
    "def play_cycloid(num_points=40, R=1.0, alpha=4.0, base_freq=220.0, note_dur=0.3):\n",
    "    x, y = cycloid_data(num_points, R=R)\n",
    "    # Just map y to pitch, ignoring x except as indexing.\n",
    "    play_sequence(y, alpha=alpha, base_freq=base_freq, note_dur=note_dur)\n",
    "    \n",
    "print(\"Cycloid functions ready.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "c1b8c110",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "### Quick Demo\n",
    "We'll do a 40-point cycloid, R=1.0, and map its y-values to pitch."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 4,
```

```
   "id": "18d9e920",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Single-voice playback done.\n"
     ]
    }
   ],
   "source": [
    "play_cycloid(num_points=40, R=1.0, alpha=5.0, base_freq=220.0, note_dur=0.25)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "5e595b8a",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
```

```
    "## 2) Simple Stock/Financial Data Example\n",
    "We'll randomly simulate some daily closing prices. Then chunk them into weekly means, and map them to a pitch line.\n",
    "\n",
    "*(In reality, you'd load real CSV data. But we'll do a quick random approach for demonstration.)*"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 5,
   "id": "cbecd15d",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Stock simulation & playback ready.\n"
     ]
    }
   ],
   "source": [
    "def simulate_stock_prices(days=60, start_price=100.0):\n",
    "    prices = [start_price]\n",
```

```
    "    for i in range(days-1):\n",
    "        change = np.random.normal(0, 2)  # daily
fluctuation\n",
    "        prices.append(max(0.0, prices[-1] + change))\n",
    "    return prices\n",
    "\n",
    "def play_stock_sonification(days=60, chunk_size=5, alpha=4.0,
base_freq=220.0, note_dur=0.3):\n",
    "    data = simulate_stock_prices(days=days)\n",
    "    chunked = chunk_data(data, chunk_size=chunk_size,
method='mean')\n",
    "    play_sequence(chunked, alpha=alpha, base_freq=base_freq,
note_dur=note_dur)\n",
    "\n",
    "print(\"Stock simulation & playback ready.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "28e6191c",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "### Quick Demo\n",
    "We'll do 60 days, chunk by 5 (representing weekly average), and
sonify. Variation might produce interesting pitch contours."
   ]
```

```
  },
  {
   "cell_type": "code",
   "execution_count": 6,
   "id": "d4874973",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Single-voice playback done.\n"
     ]
    }
   ],
   "source": [
    "play_stock_sonification(days=60, chunk_size=5, alpha=5.0, base_freq=220.0, note_dur=0.2)"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "c6ee9eb2",
   "metadata": {
    "pycharm": {
```

```
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Interactive Version\n",
    "We'll let you pick how many days, the chunk size, the pitch scaling, etc. Then re-sonify on each slider change."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 7,
   "id": "2fc3e7bb",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "8d058844deea407595efdf2f3414b8c0",
       "version_major": 2,
       "version_minor": 0
      },
      "text/plain": [
       "interactive(children=(IntSlider(value=60, description='days', max=120, min=30, step=10), IntSlider(value=5, de…"
```

```
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "@widgets.interact(\n",
    "    days=(30,120,10),\n",
    "    chunk_size=(1,10,1),\n",
    "    alpha=(1.0,8.0,1.0),\n",
    "    note_dur=(0.1,1.0,0.1)\n",
    ")\n",
    "def stock_interactive(days=60, chunk_size=5, alpha=4.0,
note_dur=0.3):\n",
    "    data = simulate_stock_prices(days=days)\n",
    "    chunked = chunk_data(data, chunk_size=chunk_size,
method='mean')\n",
    "    play_sequence(chunked, alpha=alpha, base_freq=220.0,
note_dur=note_dur)\n",
    "\n",
    "    plt.figure(figsize=(7,3))\n",
    "    plt.plot(data, 'b.-', label='Daily Prices')\n",
    "    plt.title(f\"Simulated {days} days. chunk={chunk_size},
alpha={alpha}, dur={note_dur}\")\n",
    "    plt.xlabel(\"Day\")\n",
    "    plt.ylabel(\"Price\")\n",
    "    plt.legend()\n",
    "    plt.show()\n",
    "    \n",
```

```
    "    print(\"Played interactive stock sonification.\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "2654a361-839b-47b2-b705-feaca4e88250",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Sonify Env",
   "language": "python",
   "name": "sonify_env"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
   "mimetype": "text/x-python",
```

```
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.21"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

Chapter 10: Conclusion and Future Directions

Preliminary Concepts

February 10, 2025

10.1 Introduction

We have traversed a comprehensive path in developing a **Helical Sonification System (HSS)**—from basic pitch and timbre mappings to advanced emotional cues (Chapter 7) and multi-voice scaling (Chapters 8–9). Now, in Chapter 10, we:

- Summarize the core achievements of this book,
- Reflect on **limitations** (technical, psychoacoustic, or user-related),
- Sketch **future directions**, such as VR/AR implementations, AI-driven expansions, or clinical applications.

Overview of This Chapter

1. **Major Accomplishments:** Quick bullet review of the entire HSS approach.
2. **Limitations and Considerations:** Where sonification struggles, potential pitfalls, user training, cultural factors, etc.
3. **Future Directions:** Potential for microtonal expansions, AI-based adaptive sound, VR interactions, emotional therapy, and beyond.

We'll keep the next section (Expanded Discussion) more discursive, with references to real-world prototypes and "what's next."

Chapter 10: Conclusion and Future Directions

Expanded Discussion

February 10, 2025

10.1 Core Achievements of Our Helical Sonification System

10.1.1 Summary of Chapters 1–9

Chapter 1–2 (Foundations) We introduced the rationale for **sonification** and the mathematical/music theory underpinnings:

- Helical pitch axis to merge octave cyclicity with linear frequency ascent,
- Basic psychoacoustic properties (log-frequency, scale quantization).

Chapters 3–5 (Implementation: Pitch, Timbre, and Practical Functions) We constructed the code and methodology for:

- Mapping data to pitch and simple timbre (brightness),
- Handling typical mathematical functions (e.g. damped sine),
- Avoiding "vacuum cleaner" noise via chunking or scale snapping.

Chapters 6–7 (Rhythm, Polyrhythm, Emotional Shading) We expanded into:

- Polyrhythms (3:4, multiple streams) for extra dimensional mapping,
- Emotional shading, including fear/excitement triggers (roughness, rising pitch glides, tempo above heart rate).

Chapters 8–9 (Advanced Topics, Case Studies) Finally, we tackled:

- Multi-voice setups for large data sets, real-time or VR interactions,
- Real-world examples (stock data, fractals, brachistochrone curves, game-like horror cues).

10.2 Limitations and Considerations

10.2.1 Psychoacoustic/Perceptual Complexity

Sonification is powerful but can overwhelm listeners if too many *unfamiliar* axes or voices are introduced.

- **Learning Curve**: Users may need training to interpret pitch lines, polyrhythms, or timbre shifts as meaningful data cues.
- **Individual Differences**: Some individuals have better pitch or rhythmic acuity than others.

10.2.2 Cultural and Emotional Factors

As discussed in Chapter 7, emotional connotations (major/minor, dissonance) can vary across cultures, and not everyone experiences "fear" from dissonant intervals. Additionally, too much emotional intensity might distract from the data rather than clarify it.

10.2.3 Technical Constraints

- **Real-Time Latency**: If implementing VR or direct sensor mappings, ensuring low-latency audio can be challenging.
- **Data Preprocessing**: Large or messy data demands chunking, smoothing, or dimensionality reduction to remain musically coherent.

10.3 Future Directions

10.3.1 Microtonal and Beyond-12TET Systems

We restricted ourselves mostly to 12-tone equal temperament for convenience, but microtonal expansions (19TET, 24TET, etc.) might reveal subtle data variations. Or just intonation for more "pure" intervals.

- Could be especially interesting in scientific or global contexts where standard 12TET is no longer the best representation.

10.3.2 AI and Adaptive Sonification

Machine learning models could dynamically pick the "best" scale or timbre for highlighting patterns, or adapt emotional shading based on user feedback.

- **User-Focused Approach**: If the user fails to notice a data anomaly, the system might intensify dissonance or raise pitch volume.
- **Reinforcement Learning**: The sonification engine could test different mappings and see which yields the best user comprehension.

10.3.3 VR/AR Immersive Soundscapes

Merging **3D spatial audio** with the Helical approach might let you physically "walk around" the spiral of pitches in a VR environment.

- **Gesture Controls**: wave your hand upward for higher pitch, turn your head to shift timbre axes, etc.
- **Polyrhythm in 3D Space**: place different rhythmic streams in different spatial locations, so the user can "walk" from one stream to another for more focus.

10.3.4 Clinical and Educational Uses

Biofeedback or therapy sessions might incorporate *fear triggers* or *calming cues* based on real-time physiological signals. Meanwhile, in STEAM (Science, Technology, Engineering, Arts, Mathematics) education, a dynamic sonification lab could help visually impaired students or anyone seeking a more intuitive, multi-sensory approach to mathematics and data.

10.4 Final Thoughts

The Helical Sonification System is not just about turning data into "beeps" or "sweeps." It aims to produce *music-like* structures that harness the full potential of pitch, timbre, rhythm, polyrhythm, and emotional cues. By bridging psychoacoustics, music theory, data science, and creativity, we open the door to:

- Deeper intuition for complex patterns (like fractals, high-dimensional stats, real-time sensor streams),
- Engaging, immersive experiences that might surpass purely visual graphs in certain contexts,
- A new synergy between art and science, where data *is* the music, and the listener can both enjoy it aesthetically and interpret it cognitively.

We hope this book has shown the vast possibilities and the path forward—there is still much to explore and perfect. **Sonification** remains a growing field, and with the rising power of real-time audio engines, VR/AR interfaces, and AI-based adaptations, the horizon for advanced, emotionally rich, and scientifically enlightening *musical data experiences* is brighter than ever.

chapter10_final_synergy.ipynb

```
#%% md
# Chapter 10: Final Synergy Demo

In this notebook, we combine:
- **Two voices** (Voice A, Voice B)
- An optional polyrhythm approach (3-subdivision vs. 4-subdivision triggers)
- **Emotional shading**: user picks either a major or minor scale (or toggles a 'fear factor')
- Some chunking of random data for brevity.

The idea: Show how all these elements can come together in an "everything but the kitchen sink"
demonstration. Enjoy!
#%%
import numpy as np
import sounddevice as sd
import ipywidgets as widgets
import matplotlib.pyplot as plt
print("Chapter 10 final synergy notebook loaded.")
#%% md
## Utility Functions
We'll define a polyrhythm-based "two-voice" playback that also respects a chosen scale set for each
voice (major or minor).
#%%
def snap_to_scale(val, scale=[0,2,4,5,7,9,11], semitones_per_octave=12):
    octave = int(val // semitones_per_octave)
    remainder = val - octave*semitones_per_octave
    best_note = scale[0]
    best_diff = 9999
    for s in scale:
        diff = abs(s - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = s
    return octave*semitones_per_octave + best_note

def semitone_to_freq(base_freq, semitone_offset):
    return base_freq * (2.0 ** (semitone_offset / 12.0))

def chunk_data(data, chunk_size=10, method='mean'):
    out = []
    for i in range(0, len(data), chunk_size):
        block = data[i:i+chunk_size]
        if method=='mean':
            val = np.mean(block)
        elif method=='max':
            val = np.max(block)
        else:
            val = np.mean(block)
        out.append(val)
    return out

def play_two_voice_polyrhythm(
    dataA, dataB,
    alphaA=4.0, alphaB=4.0,
    baseA=220.0, baseB=330.0,
    scaleA=[0,2,4,5,7,9,11], scaleB=[0,2,4,5,7,9,11],
    measure_duration=2.0,
    subdivA=3,
    subdivB=4,
    note_dur=0.2
):
    """
    We'll do a single measure of length measure_duration.
    Voice A triggers subdivA times, Voice B triggers subdivB times.
    Each trigger, we pick the next data point in dataA or dataB, snap pitch, and produce a short
beep.
    """
    # chunk the data if needed
    # but for now, assume dataA has subdivA length, dataB has subdivB length.
    dataA = dataA[:subdivA]
    dataB = dataB[:subdivB]

    # times
    a_times = np.linspace(0, measure_duration*(subdivA-1)/subdivA, subdivA)
    b_times = np.linspace(0, measure_duration*(subdivB-1)/subdivB, subdivB)

    events = []
```

```
    # build events for A
    for i, tA in enumerate(a_times):
        raw_semA = alphaA*dataA[i]
        snappedA = snap_to_scale(raw_semA, scale=scaleA)
        freqA = semitone_to_freq(baseA, snappedA)
        events.append((tA, 'A', freqA))
    # build events for B
    for j, tB in enumerate(b_times):
        raw_semB = alphaB*dataB[j]
        snappedB = snap_to_scale(raw_semB, scale=scaleB)
        freqB = semitone_to_freq(baseB, snappedB)
        events.append((tB, 'B', freqB))

    events.sort(key=lambda e: e[0])
    beep_sr = 44100
    last_t = 0

    for (tevent, who, freq) in events:
        wait_time = tevent - last_t
        if wait_time>0:
            sd.sleep(int(wait_time*1000))
        last_t = tevent
        wave = 0.3*np.sin(2.0*np.pi*freq*np.linspace(0, note_dur,
int(beep_sr*note_dur),endpoint=False))
        sd.play(wave, samplerate=beep_sr)
        sd.wait()

    leftover = measure_duration - last_t
    if leftover>0:
        sd.sleep(int(leftover*1000))
    print("Multi-voice polyrhythm measure done.")
#%% md
## Final Synergy Interactive Cell
We'll do the following:
- Generate random dataA, dataB (length ~8 each).
- Let user choose major or minor for each voice (valence?), or a 'fear factor' that picks a minor
scale with dissonant intervals.
- Let them set polyrhythm subdivisions (3 & 4 by default, or 4 & 5?), measure duration, note
duration, etc.

We'll do a single measure each time they change a slider.
#%%
@widgets.interact(
    scaleA=widgets.Dropdown(options={'Major':[0,2,4,5,7,9,11], 'Minor':[0,2,3,5,7,8,10], 'Fear':
[0,1,3,6,7,10,11]}, value=[0,2,4,5,7,9,11], description='Scale A'),
    scaleB=widgets.Dropdown(options={'Major':[0,2,4,5,7,9,11], 'Minor':[0,2,3,5,7,8,10], 'Fear':
[0,1,3,6,7,10,11]}, value=[0,2,4,5,7,9,11], description='Scale B'),
    alphaA=(1.0,8.0,1.0),
    alphaB=(1.0,8.0,1.0),
    subdivA=(2,6,1),
    subdivB=(2,6,1),
    measure_duration=(1.0,5.0,0.5),
    note_dur=(0.1,0.6,0.1)
)
def synergy_demo(scaleA, scaleB, alphaA=4.0, alphaB=4.0,
                 subdivA=3, subdivB=4,
                 measure_duration=2.0,
                 note_dur=0.2):
    # generate small random data for each voice
    dataA = np.random.uniform(0,5,subdivA)
    dataB = np.random.uniform(0,5,subdivB)

    print("(scaleA=", scaleA, ", scaleB=", scaleB,
          ", alphaA=", alphaA, ", alphaB=", alphaB,
          ", subdivA=", subdivA, ", subdivB=", subdivB,
          ", measure=", measure_duration, ", note_dur=", note_dur,
          ")")

    play_two_voice_polyrhythm(
        dataA, dataB,
        alphaA=alphaA, alphaB=alphaB,
        baseA=220.0, baseB=330.0,
        scaleA=scaleA, scaleB=scaleB,
        measure_duration=measure_duration,
        subdivA=subdivA,
        subdivB=subdivB,
        note_dur=note_dur
    )
    print("Synergy measure done.")
#%%
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "9ef795f4",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 10: Final Synergy Demo\n",
    "\n",
    "In this notebook, we combine:\n",
    "- **Two voices** (Voice A, Voice B)\n",
    "- An optional polyrhythm approach (3-subdivision vs. 4-subdivision triggers)\n",
    "- **Emotional shading**: user picks either a major or minor scale (or toggles a 'fear factor')\n",
    "- Some chunking of random data for brevity.\n",
    "\n",
    "The idea: Show how all these elements can come together in an \"everything but the kitchen sink\" demonstration. Enjoy!"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
   "id": "3635ca79",
   "metadata": {
```

```
   "pycharm": {
    "name": "#%%\n"
   }
  },
  "outputs": [
   {
    "name": "stdout",
    "output_type": "stream",
    "text": [
     "Chapter 10 final synergy notebook loaded.\n"
    ]
   }
  ],
  "source": [
   "import numpy as np\n",
   "import sounddevice as sd\n",
   "import ipywidgets as widgets\n",
   "import matplotlib.pyplot as plt\n",
   "print(\"Chapter 10 final synergy notebook loaded.\")"
  ]
 },
 {
  "cell_type": "markdown",
  "id": "40e135d3",
  "metadata": {
   "pycharm": {
    "name": "#%% md\n"
   }
  },
```

```
   "source": [
    "## Utility Functions\n",
    "We'll define a polyrhythm-based \"two-voice\" playback that also respects a chosen scale set for each voice (major or minor)."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "732a19ce",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
   "source": [
    "def snap_to_scale(val, scale=[0,2,4,5,7,9,11], semitones_per_octave=12):\n",
    "    octave = int(val // semitones_per_octave)\n",
    "    remainder = val - octave*semitones_per_octave\n",
    "    best_note = scale[0]\n",
    "    best_diff = 9999\n",
    "    for s in scale:\n",
    "        diff = abs(s - remainder)\n",
    "        if diff < best_diff:\n",
    "            best_diff = diff\n",
    "            best_note = s\n",
    "    return octave*semitones_per_octave + best_note\n",
```

```
"\n",
"def semitone_to_freq(base_freq, semitone_offset):\n",
"    return base_freq * (2.0 ** (semitone_offset / 12.0))\n",
"\n",
"def chunk_data(data, chunk_size=10, method='mean'):\n",
"    out = []\n",
"    for i in range(0, len(data), chunk_size):\n",
"        block = data[i:i+chunk_size]\n",
"        if method=='mean':\n",
"            val = np.mean(block)\n",
"        elif method=='max':\n",
"            val = np.max(block)\n",
"        else:\n",
"            val = np.mean(block)\n",
"        out.append(val)\n",
"    return out\n",
"\n",
"def play_two_voice_polyrhythm(\n",
"    dataA, dataB,\n",
"    alphaA=4.0, alphaB=4.0,\n",
"    baseA=220.0, baseB=330.0,\n",
"    scaleA=[0,2,4,5,7,9,11], scaleB=[0,2,4,5,7,9,11],\n",
"    measure_duration=2.0,\n",
"    subdivA=3,\n",
"    subdivB=4,\n",
"    note_dur=0.2\n",
"):\n",
"    \"\"\"\n",
"    We'll do a single measure of length measure_duration.\n",
```

```
    "    Voice A triggers subdivA times, Voice B triggers subdivB times.\n",
    "    Each trigger, we pick the next data point in dataA or dataB, snap pitch, and produce a short beep.\n",
    "    \"\"\"\n",
    "    # chunk the data if needed\n",
    "    # but for now, assume dataA has subdivA length, dataB has subdivB length.\n",
    "    dataA = dataA[:subdivA]\n",
    "    dataB = dataB[:subdivB]\n",
    "\n",
    "    # times\n",
    "    a_times = np.linspace(0, measure_duration*(subdivA-1)/subdivA, subdivA)\n",
    "    b_times = np.linspace(0, measure_duration*(subdivB-1)/subdivB, subdivB)\n",
    "\n",
    "    events = []\n",
    "    # build events for A\n",
    "    for i, tA in enumerate(a_times):\n",
    "        raw_semA = alphaA*dataA[i]\n",
    "        snappedA = snap_to_scale(raw_semA, scale=scaleA)\n",
    "        freqA = semitone_to_freq(baseA, snappedA)\n",
    "        events.append((tA, 'A', freqA))\n",
    "    # build events for B\n",
    "    for j, tB in enumerate(b_times):\n",
    "        raw_semB = alphaB*dataB[j]\n",
    "        snappedB = snap_to_scale(raw_semB, scale=scaleB)\n",
    "        freqB = semitone_to_freq(baseB, snappedB)\n",
    "        events.append((tB, 'B', freqB))\n",
```

```
    "\n",
    "    events.sort(key=lambda e: e[0])\n",
    "    beep_sr = 44100\n",
    "    last_t = 0\n",
    "\n",
    "    for (tevent, who, freq) in events:\n",
    "        wait_time = tevent - last_t\n",
    "        if wait_time>0:\n",
    "            sd.sleep(int(wait_time*1000))\n",
    "        last_t = tevent\n",
    "        wave = 0.3*np.sin(2.0*np.pi*freq*np.linspace(0,
note_dur, int(beep_sr*note_dur),endpoint=False))\n",
    "        sd.play(wave, samplerate=beep_sr)\n",
    "        sd.wait()\n",
    "\n",
    "    leftover = measure_duration - last_t\n",
    "    if leftover>0:\n",
    "        sd.sleep(int(leftover*1000))\n",
    "    print(\"Multi-voice polyrhythm measure done.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "78014e5f",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
```

```
   "source": [
    "## Final Synergy Interactive Cell\n",
    "We'll do the following:\n",
    "- Generate random dataA, dataB (length ~8 each).\n",
    "- Let user choose major or minor for each voice (valence?), or a 'fear factor' that picks a minor scale with dissonant intervals.\n",
    "- Let them set polyrhythm subdivisions (3 & 4 by default, or 4 & 5?), measure duration, note duration, etc.\n",
    "\n",
    "We'll do a single measure each time they change a slider."
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "b8fb951b",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "aed276652b8d41b0b4059992c6008a72",
       "version_major": 2,
       "version_minor": 0
      },
```

```
      "text/plain": [
       "interactive(children=(Dropdown(description='Scale A', options={'Major': [0, 2, 4, 5, 7, 9, 11], 'Minor': [0, 2…"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "@widgets.interact(\n",
    "    scaleA=widgets.Dropdown(options={'Major':[0,2,4,5,7,9,11], 'Minor':[0,2,3,5,7,8,10], 'Fear':[0,1,3,6,7,10,11]}, value=[0,2,4,5,7,9,11], description='Scale A'),\n",
    "    scaleB=widgets.Dropdown(options={'Major':[0,2,4,5,7,9,11], 'Minor':[0,2,3,5,7,8,10], 'Fear':[0,1,3,6,7,10,11]}, value=[0,2,4,5,7,9,11], description='Scale B'),\n",
    "    alphaA=(1.0,8.0,1.0),\n",
    "    alphaB=(1.0,8.0,1.0),\n",
    "    subdivA=(2,6,1),\n",
    "    subdivB=(2,6,1),\n",
    "    measure_duration=(1.0,5.0,0.5),\n",
    "    note_dur=(0.1,0.6,0.1)\n",
    ")\n",
    "def synergy_demo(scaleA, scaleB, alphaA=4.0, alphaB=4.0,\n",
    "                 subdivA=3, subdivB=4,\n",
    "                 measure_duration=2.0,\n",
    "                 note_dur=0.2):\n",
    "    # generate small random data for each voice\n",
    "    dataA = np.random.uniform(0,5,subdivA)\n",
```

```
    "    dataB = np.random.uniform(0,5,subdivB)\n",
    "    \n",
    "    print(\"(scaleA=\", scaleA, \", scaleB=\", scaleB,\n",
    "          \", alphaA=\", alphaA, \", alphaB=\", alphaB,\n",
    "          \", subdivA=\", subdivA, \", subdivB=\", subdivB,\n",
    "          \", measure=\", measure_duration, \", note_dur=\",
note_dur,\n",
    "          \")\")\n",
    "    \n",
    "    play_two_voice_polyrhythm(\n",
    "        dataA, dataB,\n",
    "        alphaA=alphaA, alphaB=alphaB,\n",
    "        baseA=220.0, baseB=330.0,\n",
    "        scaleA=scaleA, scaleB=scaleB,\n",
    "        measure_duration=measure_duration,\n",
    "        subdivA=subdivA,\n",
    "        subdivB=subdivB,\n",
    "        note_dur=note_dur\n",
    "    )\n",
    "    print(\"Synergy measure done.\")"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "1e059c66-802d-4a3c-8ead-816242b52282",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
```

```
      }
     },
     "outputs": [],
     "source": []
    }
   ],
   "metadata": {
    "kernelspec": {
     "display_name": "Sonify Env",
     "language": "python",
     "name": "sonify_env"
    },
    "language_info": {
     "codemirror_mode": {
      "name": "ipython",
      "version": 3
     },
     "file_extension": ".py",
     "mimetype": "text/x-python",
     "name": "python",
     "nbconvert_exporter": "python",
     "pygments_lexer": "ipython3",
     "version": "3.9.21"
    }
   },
   "nbformat": 4,
   "nbformat_minor": 5
  }
```

Chapter10_Final_Synergy_Debug.ipynb

```
#%% md
# Chapter 10: Final Synergy (Debug Version)

This notebook:
1. Tests audio with a **simple beep** (Cell 2).
2. Defines our **polyrhythm synergy** code with **debug prints** and **longer note durations**.
3. Provides an **interactive** set of sliders to control scale, pitch scale, measure duration, etc.

Copy exactly from the first `{` to the last `}` into a file named
`Chapter10_Final_Synergy_Debug.ipynb`. Then open in Jupyter. No extra backticks or lines.
If you see frequency values printed but still hear silence, check volume settings or `sounddevice`
environment.

#%%
# Cell 2: Simple beep test
import numpy as np
import sounddevice as sd

def test_beep(duration=1.0, freq=440.0):
    sr = 44100
    t = np.linspace(0, duration, int(sr*duration), endpoint=False)
    wave = 0.3 * np.sin(2*np.pi*freq * t)
    sd.play(wave, samplerate=sr)
    sd.wait()
    print(f"Test beep at {freq} Hz for {duration} s done.")

# Let's do a 1-second beep at 440 Hz.
test_beep()
#%% md
## Cell 3: Synergy polyrhythm code
We'll define a polyrhythm function that triggers two voices (A and B) in one measure, subdividing
time.
We also add debug prints of frequency.

#%%
import ipywidgets as widgets
import matplotlib.pyplot as plt

def snap_to_scale(val, scale=[0,2,4,5,7,9,11], semitones_per_octave=12):
    octave = int(val // semitones_per_octave)
    remainder = val - octave*semitones_per_octave
    best_note = scale[0]
    best_diff = 9999
    for s in scale:
        diff = abs(s - remainder)
        if diff < best_diff:
            best_diff = diff
            best_note = s
    return octave*semitones_per_octave + best_note

def semitone_to_freq(base_freq, semitone_offset):
    return base_freq * (2.0 ** (semitone_offset / 12.0))

def play_polyrhythm_synergy(
    dataA, dataB,
    alphaA=4.0, alphaB=4.0,
    baseA=220.0, baseB=330.0,
    scaleA=[0,2,4,5,7,9,11], scaleB=[0,2,4,5,7,9,11],
    measure_duration=2.0,
    subdivA=3,
    subdivB=4,
    note_dur=0.5
):
    """
    We'll do a single measure of length measure_duration.
    Voice A triggers subdivA times, voice B triggers subdivB times.
    Each event is a short beep of length note_dur.
    dataA and dataB must have at least subdivA/subdivB length.
    Debug prints show time/freq.
    """
    dataA = dataA[:subdivA]
    dataB = dataB[:subdivB]

    a_times = np.linspace(0, measure_duration*(subdivA-1)/subdivA, subdivA)
    b_times = np.linspace(0, measure_duration*(subdivB-1)/subdivB, subdivB)

    events = []
```

```
    # build A events
    for i, tA in enumerate(a_times):
        raw_semA = alphaA*dataA[i]
        snappedA = snap_to_scale(raw_semA, scale=scaleA)
        freqA = semitone_to_freq(baseA, snappedA)
        events.append((tA, 'A', freqA))
    # build B events
    for j, tB in enumerate(b_times):
        raw_semB = alphaB*dataB[j]
        snappedB = snap_to_scale(raw_semB, scale=scaleB)
        freqB = semitone_to_freq(baseB, snappedB)
        events.append((tB, 'B', freqB))

    events.sort(key=lambda e: e[0])
    beep_sr = 44100
    last_t = 0

    for (tevent, who, freq) in events:
        wait_time = tevent - last_t
        if wait_time>0:
            sd.sleep(int(wait_time*1000))
        last_t = tevent
        print(f"Time={tevent:.2f}, Voice={who}, Freq={freq:.2f} Hz")
        wave = 0.3*np.sin(2.0*np.pi*freq*np.linspace(0, note_dur,
int(beep_sr*note_dur),endpoint=False))
        sd.play(wave, samplerate=beep_sr)
        sd.wait()

    leftover = measure_duration - last_t
    if leftover>0:
        sd.sleep(int(leftover*1000))
    print("Polyrhythm synergy measure done.")

print("Polyrhythm synergy function defined.")
#%% md
## Cell 4: Interactive Sliders
We generate random data arrays `dataA` and `dataB` each time you adjust a slider. Then we do a
single measure of polyrhythm synergy, with half-second notes. Should be very noticeable.

#%%
@widgets.interact(
    scaleA=widgets.Dropdown(options={'Major':[0,2,4,5,7,9,11], 'Minor':[0,2,3,5,7,8,10], 'Fear':
[0,1,3,6,7,10,11]}, value=[0,2,4,5,7,9,11], description='Scale A'),
    scaleB=widgets.Dropdown(options={'Major':[0,2,4,5,7,9,11], 'Minor':[0,2,3,5,7,8,10], 'Fear':
[0,1,3,6,7,10,11]}, value=[0,2,4,5,7,9,11], description='Scale B'),
    alphaA=(1.0,8.0,1.0),
    alphaB=(1.0,8.0,1.0),
    subdivA=(2,6,1),
    subdivB=(2,6,1),
    measure_duration=(1.0,5.0,0.5),
    note_dur=(0.2,1.0,0.1)
)
def synergy_interactive(
    scaleA, scaleB,
    alphaA=6.0, alphaB=6.0,
    subdivA=3, subdivB=4,
    measure_duration=2.0,
    note_dur=0.5
):
    # generate random data each time, length >= subdivA, subdivB
    dataA = np.random.uniform(0,5,subdivA)
    dataB = np.random.uniform(0,5,subdivB)

    print(f"Chosen scales, alphaA={alphaA}, alphaB={alphaB}, subdivA={subdivA}, subdivB={subdivB},
measure={measure_duration}, note_dur={note_dur}")
    print("Data A:", dataA)
    print("Data B:", dataB)

    play_polyrhythm_synergy(
        dataA, dataB,
        alphaA=alphaA, alphaB=alphaB,
        baseA=220.0, baseB=330.0,
        scaleA=scaleA, scaleB=scaleB,
        measure_duration=measure_duration,
        subdivA=subdivA,
        subdivB=subdivB,
        note_dur=note_dur
    )
    print("Done synergy interactive call.")

#%%
```

```
{
 "cells": [
  {
   "cell_type": "markdown",
   "id": "08401286",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "# Chapter 10: Final Synergy (Debug Version)\n",
    "\n",
    "This notebook:\n",
    "1. Tests audio with a **simple beep** (Cell 2).\n",
    "2. Defines our **polyrhythm synergy** code with **debug prints** and **longer note durations**.\n",
    "3. Provides an **interactive** set of sliders to control scale, pitch scale, measure duration, etc.\n",
    "\n",
    "Copy exactly from the first `{` to the last `}` into a file named `Chapter10_Final_Synergy_Debug.ipynb`. Then open in Jupyter. No extra backticks or lines.\n",
    "If you see frequency values printed but still hear silence, check volume settings or `sounddevice` environment.\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 1,
```

```
   "id": "76e4a2de",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Test beep at 440.0 Hz for 1.0 s done.\n"
     ]
    }
   ],
   "source": [
    "# Cell 2: Simple beep test\n",
    "import numpy as np\n",
    "import sounddevice as sd\n",
    "\n",
    "def test_beep(duration=1.0, freq=440.0):\n",
    "    sr = 44100\n",
    "    t = np.linspace(0, duration, int(sr*duration),
endpoint=False)\n",
    "    wave = 0.3 * np.sin(2*np.pi*freq * t)\n",
    "    sd.play(wave, samplerate=sr)\n",
    "    sd.wait()\n",
    "    print(f\"Test beep at {freq} Hz for {duration} s
done.\")\n",
```

```
    "\n",
    "# Let's do a 1-second beep at 440 Hz.\n",
    "test_beep()"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "97351922",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Cell 3: Synergy polyrhythm code\n",
    "We'll define a polyrhythm function that triggers two voices (A and B) in one measure, subdividing time.\n",
    "We also add debug prints of frequency.\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 2,
   "id": "f85abe78",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
```

```
   "outputs": [
    {
     "name": "stdout",
     "output_type": "stream",
     "text": [
      "Polyrhythm synergy function defined.\n"
     ]
    }
   ],
   "source": [
    "import ipywidgets as widgets\n",
    "import matplotlib.pyplot as plt\n",
    "\n",
    "def snap_to_scale(val, scale=[0,2,4,5,7,9,11], 
semitones_per_octave=12):\n",
    "    octave = int(val // semitones_per_octave)\n",
    "    remainder = val - octave*semitones_per_octave\n",
    "    best_note = scale[0]\n",
    "    best_diff = 9999\n",
    "    for s in scale:\n",
    "        diff = abs(s - remainder)\n",
    "        if diff < best_diff:\n",
    "            best_diff = diff\n",
    "            best_note = s\n",
    "    return octave*semitones_per_octave + best_note\n",
    "\n",
    "def semitone_to_freq(base_freq, semitone_offset):\n",
    "    return base_freq * (2.0 ** (semitone_offset / 12.0))\n",
    "\n",
```

```
    "def play_polyrhythm_synergy(\n",
    "    dataA, dataB,\n",
    "    alphaA=4.0, alphaB=4.0,\n",
    "    baseA=220.0, baseB=330.0,\n",
    "    scaleA=[0,2,4,5,7,9,11], scaleB=[0,2,4,5,7,9,11],\n",
    "    measure_duration=2.0,\n",
    "    subdivA=3,\n",
    "    subdivB=4,\n",
    "    note_dur=0.5\n",
    "):\n",
    "    \"\"\"\n",
    "    We'll do a single measure of length measure_duration.\n",
    "    Voice A triggers subdivA times, voice B triggers subdivB
times.\n",
    "    Each event is a short beep of length note_dur.\n",
    "    dataA and dataB must have at least subdivA/subdivB
length.\n",
    "    Debug prints show time/freq.\n",
    "    \"\"\"\n",
    "    dataA = dataA[:subdivA]\n",
    "    dataB = dataB[:subdivB]\n",
    "\n",
    "    a_times = np.linspace(0, measure_duration*(subdivA-
1)/subdivA, subdivA)\n",
    "    b_times = np.linspace(0, measure_duration*(subdivB-
1)/subdivB, subdivB)\n",
    "\n",
    "    events = []\n",
    "    # build A events\n",
    "    for i, tA in enumerate(a_times):\n",
```

```
    "        raw_semA = alphaA*dataA[i]\n",
    "        snappedA = snap_to_scale(raw_semA, scale=scaleA)\n",
    "        freqA = semitone_to_freq(baseA, snappedA)\n",
    "        events.append((tA, 'A', freqA))\n",
    "    # build B events\n",
    "    for j, tB in enumerate(b_times):\n",
    "        raw_semB = alphaB*dataB[j]\n",
    "        snappedB = snap_to_scale(raw_semB, scale=scaleB)\n",
    "        freqB = semitone_to_freq(baseB, snappedB)\n",
    "        events.append((tB, 'B', freqB))\n",
    "\n",
    "    events.sort(key=lambda e: e[0])\n",
    "    beep_sr = 44100\n",
    "    last_t = 0\n",
    "\n",
    "    for (tevent, who, freq) in events:\n",
    "        wait_time = tevent - last_t\n",
    "        if wait_time>0:\n",
    "            sd.sleep(int(wait_time*1000))\n",
    "        last_t = tevent\n",
    "        print(f\"Time={tevent:.2f}, Voice={who}, 
Freq={freq:.2f} Hz\")\n",
    "        wave = 0.3*np.sin(2.0*np.pi*freq*np.linspace(0, 
note_dur, int(beep_sr*note_dur),endpoint=False))\n",
    "        sd.play(wave, samplerate=beep_sr)\n",
    "        sd.wait()\n",
    "\n",
    "    leftover = measure_duration - last_t\n",
    "    if leftover>0:\n",
```

```
    "        sd.sleep(int(leftover*1000))\n",
    "    print(\"Polyrhythm synergy measure done.\")\n",
    "\n",
    "print(\"Polyrhythm synergy function defined.\")"
   ]
  },
  {
   "cell_type": "markdown",
   "id": "12ed20c5",
   "metadata": {
    "pycharm": {
     "name": "#%% md\n"
    }
   },
   "source": [
    "## Cell 4: Interactive Sliders\n",
    "We generate random data arrays `dataA` and `dataB` each time you adjust a slider. Then we do a single measure of polyrhythm synergy, with half-second notes. Should be very noticeable.\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": 3,
   "id": "0f259edc",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
```

```
   },
   "outputs": [
    {
     "data": {
      "application/vnd.jupyter.widget-view+json": {
       "model_id": "bdd35472ef864154bff4ddd6224c95ec",
       "version_major": 2,
       "version_minor": 0
      },
      "text/plain": [
       "interactive(children=(Dropdown(description='Scale A', options={'Major': [0, 2, 4, 5, 7, 9, 11], 'Minor': [0, 2…"
      ]
     },
     "metadata": {},
     "output_type": "display_data"
    }
   ],
   "source": [
    "@widgets.interact(\n",
    "    scaleA=widgets.Dropdown(options={'Major':[0,2,4,5,7,9,11], 'Minor':[0,2,3,5,7,8,10], 'Fear':[0,1,3,6,7,10,11]}, value=[0,2,4,5,7,9,11], description='Scale A'),\n",
    "    scaleB=widgets.Dropdown(options={'Major':[0,2,4,5,7,9,11], 'Minor':[0,2,3,5,7,8,10], 'Fear':[0,1,3,6,7,10,11]}, value=[0,2,4,5,7,9,11], description='Scale B'),\n",
    "    alphaA=(1.0,8.0,1.0),\n",
    "    alphaB=(1.0,8.0,1.0),\n",
    "    subdivA=(2,6,1),\n",
    "    subdivB=(2,6,1),\n",
```

```
    "    measure_duration=(1.0,5.0,0.5),\n",
    "    note_dur=(0.2,1.0,0.1)\n",
    ")\n",
    "def synergy_interactive(\n",
    "    scaleA, scaleB,\n",
    "    alphaA=6.0, alphaB=6.0,\n",
    "    subdivA=3, subdivB=4,\n",
    "    measure_duration=2.0,\n",
    "    note_dur=0.5\n",
    "):\n",
    "    # generate random data each time, length >= subdivA,
subdivB\n",
    "    dataA = np.random.uniform(0,5,subdivA)\n",
    "    dataB = np.random.uniform(0,5,subdivB)\n",
    "\n",
    "    print(f\"Chosen scales, alphaA={alphaA}, alphaB={alphaB},
subdivA={subdivA}, subdivB={subdivB}, measure={measure_duration},
note_dur={note_dur}\")\n",
    "    print(\"Data A:\", dataA)\n",
    "    print(\"Data B:\", dataB)\n",
    "\n",
    "    play_polyrhythm_synergy(\n",
    "        dataA, dataB,\n",
    "        alphaA=alphaA, alphaB=alphaB,\n",
    "        baseA=220.0, baseB=330.0,\n",
    "        scaleA=scaleA, scaleB=scaleB,\n",
    "        measure_duration=measure_duration,\n",
    "        subdivA=subdivA,\n",
    "        subdivB=subdivB,\n",
    "        note_dur=note_dur\n",
```

```
    "    )\n",
    "    print(\"Done synergy interactive call.\")\n"
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "id": "d5d7c2ac-dcce-4a19-9579-4363f4f5a822",
   "metadata": {
    "pycharm": {
     "name": "#%%\n"
    }
   },
   "outputs": [],
   "source": []
  }
 ],
 "metadata": {
  "kernelspec": {
   "display_name": "Sonify Env",
   "language": "python",
   "name": "sonify_env"
  },
  "language_info": {
   "codemirror_mode": {
    "name": "ipython",
    "version": 3
   },
   "file_extension": ".py",
```

```
   "mimetype": "text/x-python",
   "name": "python",
   "nbconvert_exporter": "python",
   "pygments_lexer": "ipython3",
   "version": "3.9.21"
  }
 },
 "nbformat": 4,
 "nbformat_minor": 5
}
```

Epilogue

And so we come full circle. We began with the notion that Cartesian coordinates revolutionized our visual grasp of equations, forging new frontiers in calculus and modern science. We end with a realization that **sonic coordinates** might open an equally transformative path. Over these chapters, we've constructed a Helical Sonification System, weaving together pitch arcs, timbral shifts, rhythmic structures, polyrhythms, and even emotional cues like fear or tension. We explored how these elements, grounded in both psychoacoustics and music theory, can enliven raw data and turn it into a **living, breathing**, or perhaps "singing," entity.

Yet what we've shared is only the beginning. Much as Descartes' geometry needed Leibniz' and Newton's calculus to reveal its full potential, our helical system could benefit from future leaps: advanced VR interfaces, AI-driven emotional shading, or real-time adaptive systems that respond to user feedback. Perhaps a day will come when scientists routinely "jam" with their data, hearing gravitational waves or stock market fluctuations as spontaneously as we now click on a spreadsheet. Perhaps educators will use multi-voice sonification in a VR lab to let students **feel** fractal recursions or **hear** chaotic signals—like turning mathematics class into an immersive concert.

We also find ourselves at a threshold where the lines between "art" and "science" blur. Throughout history, the best mathematicians often likened their discipline to music, while many composers used mathematical structures for inspiration. Our approach simply intensifies that synergy, bridging data analysis and creative expression. The same polyrhythms that highlight correlations in a climate dataset might serve as the rhythmic drive in a new avant-garde composition. The same "fear triggers" that help a medical researcher detect anomalies in EEG signals might also be the backbone of a horror game's dynamic soundtrack. We stand at the dawn of a new **synthetic** world—where numbers sing, music calculates, and the boundary between them dissolves in a swirl of pitch arcs and timbral illusions.

If there's one final message to take away from this book, it's a sense of curiosity. Don't be afraid to experiment with "weird" sonic mappings, or to let data sculpt your composition, or to let your VR gestures modulate a multi-voice sonification. In stepping beyond the purely visual or textual analysis that's dominated so much of mathematics and data science, we give ourselves permission to stumble on unexpected insights. And

in bridging emotion—fear, excitement, awe—we remind ourselves that data is more than just knowledge; it can **feel** profoundly human when presented through the universal language of sound.

We hope you'll continue this sonic journey, forging new software, new art, new research, and new experiences. As you close these pages, may you do so with your ears wide open, eager to hear the hidden patterns of the universe. May you remember that once, a brilliant insight let us visualize equations—and now, we stand on the cusp of letting equations and data **sing**. The stage is set. The next verse is yours to write.